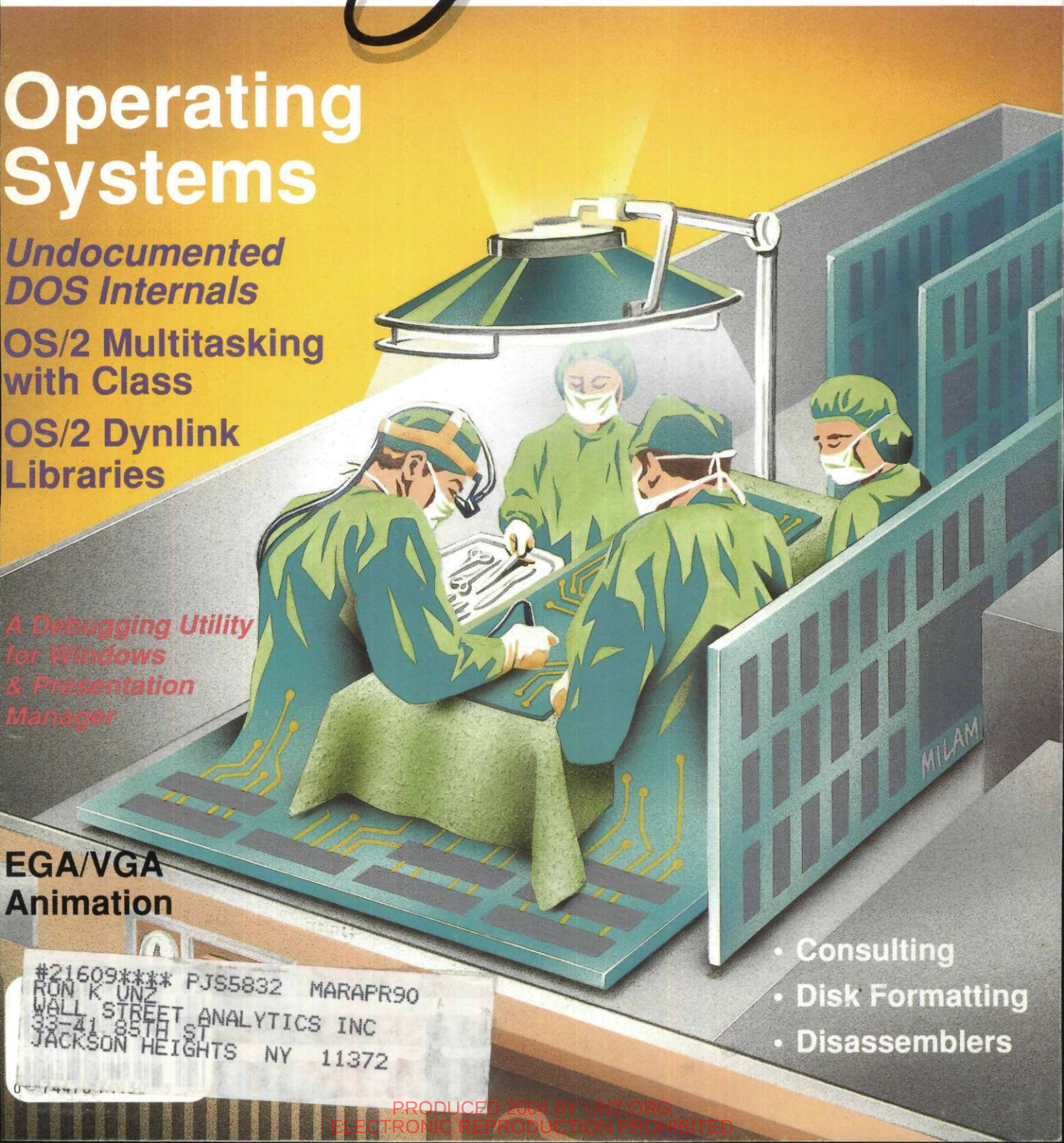
OS/2 FEATURE

The Resource Journal for IBM Micro Programmers

PROGRAMMER'S Journal

MARCH/APRIL 1989
Volume 7.2 $4.95
Canada $6.25



Operating Systems

Undocumented DOS Internals

OS/2 Multitasking with Class

OS/2 Dynlink Libraries

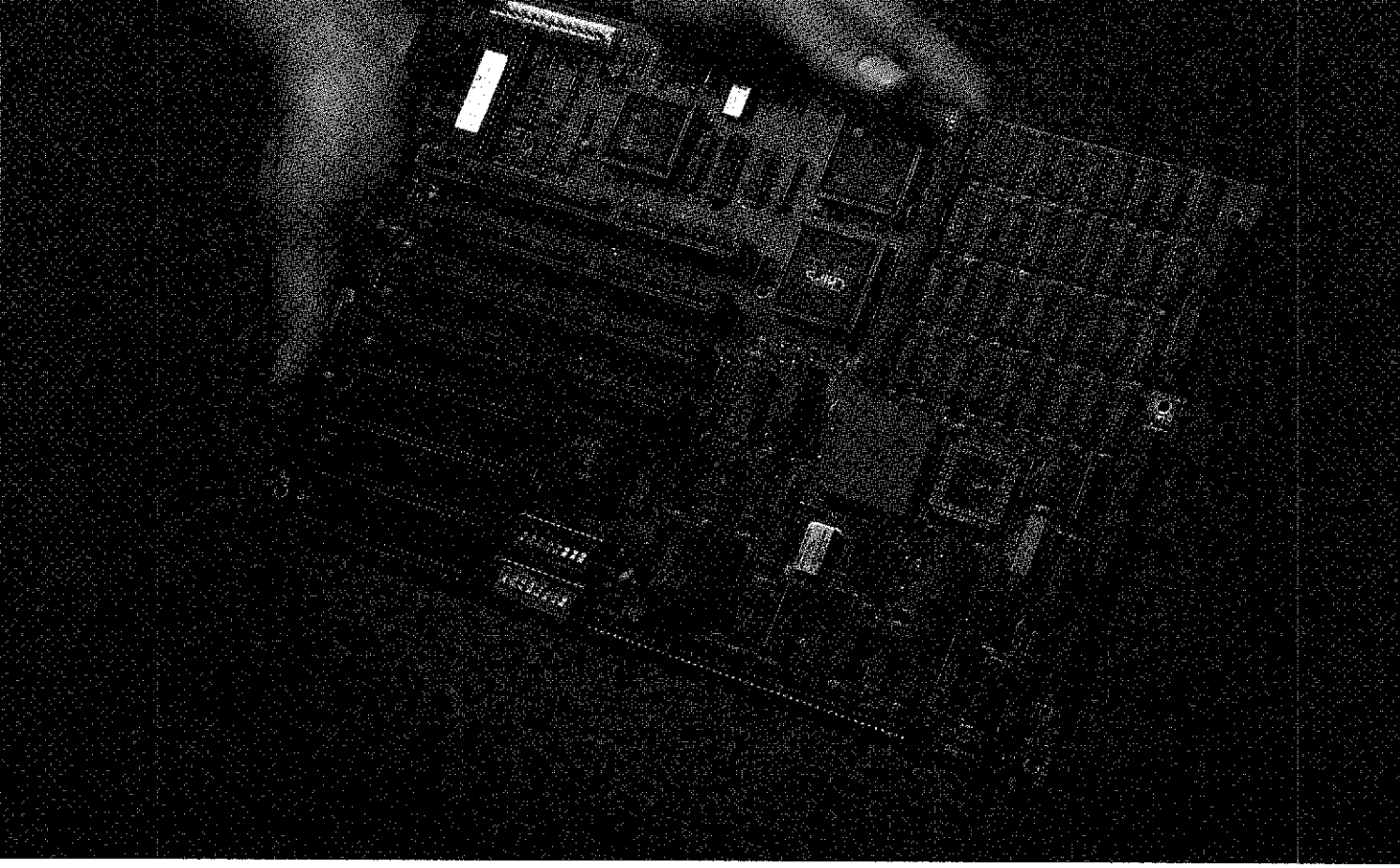
A Debugging Utility for Windows & Presentation Manager

EGA/VGA Animation

- Consulting
- Disk Formatting
- Disassemblers

#21609*** PJS5832 MARAPR90
RON K UNZ
WALL STREET ANALYTICS INC
33-41 85TH ST
JACKSON HEIGHTS NY 11372

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



CONVERT YOUR PC, XT OR AT INTO A HIGHER FORM OF LIFE!

Don't let your PC give up the ghost — Hauppauge has arrived with a new spark of life: the 386 MotherBoard.™ Far more advanced than an accelerator card, our line of MotherBoards grace your PC, PC/XT, PC/AT or compatible with speeds equal to the IBM PS2 Model 80. And faster. Because we've built in 1 Megabyte of high speed RAM and a 387 math coprocessor socket for speeds that will make your PC glad to be alive.

UNIX/386 and OS/2 Compatible. To ensure a long, fruitful life, our 386 MotherBoards are compatible with the PC/AT (BIOS and I/O) — so you can run the new generation of DOS, OS/2. Our Boards also run Windows/386, UNIX V, PC-MOS/386, and XENIX/386. For *more* power, you'll find 16-bit expansion slots that accommodate the latest I/O expansion cards. Plus, we can add up to 15 Megabytes of extended memory. No 386 accelerator card can give you so much versatility. **Only our 386 MotherBoards give your PC s an unlimited future!**

The Critics Applaud! *PC Magazine* awarded our Board "The Editor's Choice" for 386 Replacement Boards. *PC World* called it "the Upgrade Product of the Year." And *PC Computing* said: "The 386 MotherBoard is exceptionally compatible with existing IBM applications and peripherals — it will even run IBM's own version of OS/2."

Technical Features ▪ 16 or 20 MHz 80386 ▪ 1 Megabyte of 100 nsec 4-way interleaved RAM ▪ PC/AT compatible I/O and BIOS for support of OS/2 ▪ Six 8-bit expansion slots and two 16-bit expansion slots (four on 386 MotherBoard for AT) ▪ One 32-bit expansion slot for up to 15 Megabytes of high speed memory.

386 MotherBoard for the PC, PC/XT, or Compaq portable	$1695
386 MotherBoard for the PC/AT	$1795
386 MotherBoard/20 MHz for PC/AT	$2195
32-bit RAM Board	from $195
16 MHz 80387 math coprocessor	$695
16-bit combination hard disk/floppy disk controller	$245

prices subject to change

Our 386 MotherBoard will give your computer a big step up. For more information, call: **1 (800) 443-6284**. In New York: **(516) 434-1600**. All our products are available at your local computer dealer. (Ask about our GSA pricing!)

Hauppauge Computer Works, Inc.
175 Commerce Drive,
Hauppauge, New York 11788.
(In West Germany call: 0211-594-320)

Hauppauge!

[illegible] 386: Intel Corp.; Windows/386: Microsoft Corp.

Number 122 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Simply the BEST C and Pascal on AT, 386, Sun, Apollo, RT, VAX, 370

"The most rock-solid C compiler in the industry. Superb technical support and portability. Superior code generated."
Gordon Eubanks, Symantec — Q&A (386).

"It simply works, with no trouble, no chasing strange bugs, and excellent warning and error messages ... a professional product."
Robert Lerche, Bay Partners.

"For large-scale software development, the highest quality C compiler available on the market today. Pragmas are great. Quality of support is exceptional." **Randy Neilsen, Ansa—Paradox** (DOS,OS/2).

"15% smaller and 15% faster than Lattice C."
Robert Wenig, Autodesk.

"Our software is running anywhere from 30 to 50% faster than when compiled under Lattice." **David Marcus, Micronetics.**

"We switched from Lattice due to a 10% reduction in code size. The compiler is very stable." **Lee Lorenzen, Ventura Software — Ventura Publisher, marketted by Xerox Corp.**

"Best quality emitted code by any compiler I've encountered. Often amazing." **Bill Ferguson, Fox Software — FoxBase (386).**

"Messages sometimes pointed out type mismatches, incorrect-length argument lists, and uninitialized variables that had been undetected for years [in 4.x bsd]." **Larry Breed, IBM ACIS [RT PC].**

"Diagnostics turned up bugs missed by other compilers. Rapid bug fixes by technical support, someone who knew what he was talking about. 80386 code is well optimized."
Tim Addison, Logistics Data Systems.

"386 protected mode support is fantastic, especially the access to large amounts of memory. It's mainframe compute power on a PC." **Dan Eggleston, Viewlogic.**

"The preprocessor supplied with Professional Pascal is quite useful. The code quality and control over segmentation and memory models are superior to MS Pascal." **Bob Wallace, QuickSoft.**

Check Out These Reviews

- **High C ™:**

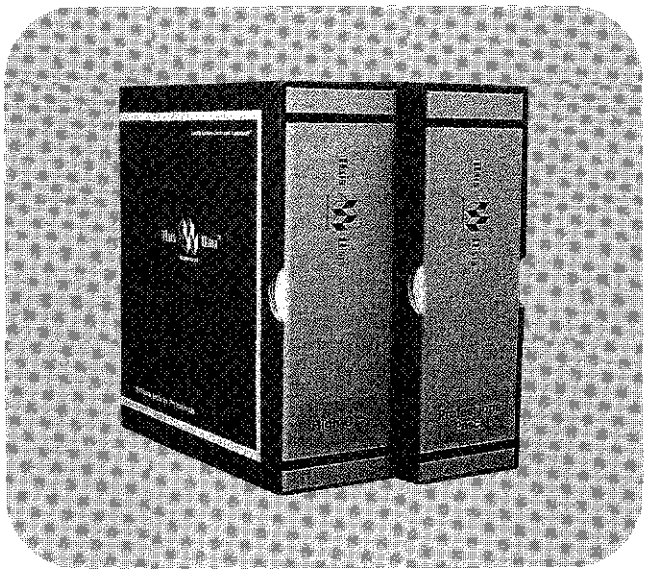
Computer Language	February 1986, '87	
Dr. Dobb's Journal	August 1986	
PC Magazine	Jan. 27, 1987	(80386 version)
Dr. Dobb's Journal	July 1987	(80386 version)
BYTE Magazine	November 1987	(80386 version)

- **Professional Pascal ™:**

PC Magazine	Dec. 29, 1985	
Computer Language	May 1986	
PC Tech Journal	July 1986	
Journal of Pascal, Ada, & Modula-2	Nov.-Dec. 1986	
BYTE Magazine	Dec. '86, June '87	(80386 version)

Why MetaWare compilers

- They are specifically designed for serious software developers.
- They are reliable and robust: they don't break at every turn.
- Their generated code is the best, or near best, on each architecture.
- Their superior diagnostic messages help you produce better products more quickly.
- Your source can be ported with ease to the most popular systems.
- You can link mixed-language modules from our compilers, others
- You can benefit from high-level, personal technical support.
- You can take advantage of the latest ANSI C extensions, and/or extensive Pascal extensions. **High C** has been tracking the ANSI Standard for two years; **Professional Pascal** will soon have a VS Pascal compatibility switch and several Apollo Pascal ext'ns.
- You can take advantage of the latest 387 and Weitek 1167 support — we have the only compilers with Weitek real mode support.



Power Tools for Power Users

Ashton-Tate: dBase III Plus, MultiMate; Autodesk: AUTOCAD, AUTOSKETCH (8087, '387, Weitek); Boeing Computer Services (Sun); CASE Technology (Sun); CAD/CAM giant Daisy ('86, '386, VAX); Deloitte Haskins & Sells; Digital Research: FlexOS; GE; IBM: 4.3/RT, 4680 OS; Lifetree Software (Pascal): Volkswriter Deluxe, GEM-Write; Lugaru: Epsilon; NYU: Ada-Ed cmplr; Semantec: Q&A; Sky Computers; ... (Product names are trademarks of the companies indicated.)

Industrial-Strength

MetaWare C and Pascal compilers are designed for professional software developers. These tools are loaded with options to control them for special purposes. You can adjust the space-time trade-off in code quality. You can adjust external naming conventions to agree with linkers and operating systems. You can specify segment names for segmented architectures, and to help place code or data in particular places for embedded applications. You can select from five memory models for the 8086 family. And on and on.

A Partial List of Optimizations

Common subexpression and dead-code elimination, retention and reuse of register contents, jump-instruction size minimization, tail merging (cross jumping), constant folding, short-circuit evaluation of Boolean expressions, strength reductions, fast procedure calls, automatic mapping of variables to registers (where advantageous), ...

"Platform" — Code Quality

Sun,Apollo,SGI	— 18%, 3%, 26% > resident compiler (Dhrystone).
PC: DOS, OS/2	— 3-10% > Microsoft C; 30% > MS Pascal, LatticeC.
386 32-bit DOS	— no competitors, since November, 1986.
286, 386 UNIX	— 66% better than pcc (Dhrystone, 386).
VAX VMS	— ≈ DEC's excellent C and Pascal; better features.
VAX Ultrix	— 19% > pcc (Dhrystone); much > Berkeley Pascal.
RT PC/4.3bsd	— 89% > the original port of pcc (Dhrystone).
370 CMS,UNIX	— much better than any C, and VS Pascal.
AMD 29000	— >40,000 Dhrystones! Available in Q2, cross.

(408) 429-6382, telex 493-0879. ***Since 1979.***

MetaWare™ INCORPORATED

903 Pacific Avenue, Suite 201 • Santa Cruz, CA 95060-4429

The Clear Choice for Large Programming Projects — PC Tech J.

© 1987 MetaWare Incorporated. MetaWare, High C, Professional Pascal, and DOS Helper are trademarks of MetaWare Incorporated. Others and their owners are duely respected.

Number 45 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

The Resource Journal for IBM Micro Programmers
PROGRAMMER'S Journal
Volume 7.2
March/April 1989

Contents

4 Notes from the Editor

6 Letters
Ron Schroeder, Howard Rensin, Dalton Orr, and Christopher A. Howard

11 Consulting: Reselling *by Jake Richter*
How to become a computer products reseller

18 On Graphics: EGA and VGA Animation *by Michael Abrash*
Using the EGA/VGA bit-plane architecture and color palette for animation

32 DOS: Undocumented DOS Internals *by Raymond J. Michels*
A tour of undocumented DOS features—processes, memory management, files, and devices

38 386: 386 Now *by M. Steven Baker*
Taking advantage of fast 80386 block moves under DOS

40 OS/2: OS/2 Dynamic Link Libraries *by Ray Duncan*
Adapted from *Advanced OS/2 Programming* from Microsoft Press, 1989

48 C++: OS/2 Multitasking with Class *by Vaughn Vernon*
Some classes for OS/2 threads and semaphores

56 C: A Simple Debugging Utility for Presentation Manager and Windows *by William S. Hall*
Our resident PM/Windows expert shows how to write to a debugging window

74 C: The Proposed ANSI C Language Standard *by Rex Jaeschke*
Rex looks at type modifiers, structure expressions, and MIX Power C V1.2

78 Pascal: Disk Formatting *by John Newlin*
System programming to deal with various floppy disks formats and formatting

86 Miscellaneous *by M. Steven Baker*
Windowing and Graphics
XVT 1.2 (Advanced Programming Institute)
TIGA-340 and TI Development Tools
Disassemblers
SOURCER (V Communications)
snOOp II (TriDOS Software)
MD86 (C. C. Software)
Books
The 80386 Book by Ross Nelson (Microsoft Press)

95 PJ Billboard

96 Advertiser Index

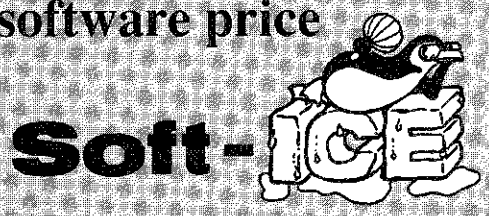
Hackers at work? No. Simply our way of illustrating how operating systems are at the heart of it all. By the way, the patient lived.

Cover concept:
Bob Wanetick
Artist:
Larry Milam
Art Director:
Ralph Bentley

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

SERIOUS DEBUGGING *at a* REASONABLE PRICE

All the speed and power of a hardware-assisted debugger at a software price



Hardware-level break points
REAL-TIME break points on memory locations, memory ranges, execution, I/O ports, hardware and software interrupts. More powerful break points than ANY software-only debugger on the market. Soft-ICE gives you the power of an in-circuit emulator on your desk.

Break out of hung programs
With a keystroke - no external switch necessary. Even with interrupts disabled.

Breaks the 640K barrier
Soft-ICE uses ZERO bytes of memory in the first 1MB of address space. This is especially useful for those subtle bugs that change when the starting address of your code changes. With Soft-ICE your code executes at the same address whether the debugger is loaded or not.

Works with your favorite debugger
Soft-ICE can be used as a stand-alone debugger or it can add its powerful break points to the software debugger you already use. You can continue to use your favorite debugger until you require Soft-ICE. Simply pop up the Soft-ICE window to set powerful real-time break points. When a break point is reached, your debugger will be activated.

Solve tough systems problems too
Soft-ICE is ideal for debugging TSRs, interrupt handlers, self booting programs, DOS loadable device drivers, non-DOS operating systems, and debugging within DOS & BIOS. Soft-ICE is also great for firmware development because Soft-ICE's break points work in ROM.

How Soft-ICE Works
Soft-ICE uses the power of the 80386 to surround your program in a virtual machine. This gives you complete control of the DOS environment, while Soft-ICE runs safely in protected mode. Soft-ICE uses 80386 protected mode features, such as paging, I/O privilege level, and break point registers, to provide real-time hardware-level break points.

"Soft-ICE is a product any MS-DOS developer serious enough to own a 386 machine should have."
Dr. Dobb's Journal — May 1988

RUN CODEVIEW IN ONLY 8K!

CodeView is a great integrated debugger, but it uses over 200K of conventional memory. MagicCV uses advanced features of the 80386 microprocessor to load CodeView and symbols in extended memory. This allows MagicCV to run CodeView using less than 8K of conventional memory on your 80386 PC.

Don't let 640K be your limit!
If you are closing in on the 640K limit and would like the power of CodeView, MagicCV is for you.

Don't let the debugger hide the bug!
Even if you're not closing in on the 640K limit, running CodeView with MagicCV makes your debugging environment much closer to the end user's program environment. You can use CodeView to locate subtle bugs that only occur when there is plenty of free memory, or those difficult bugs that only occur when your program is running with a couple of TSRs loaded.

How MagicCV works
MagicCV uses the 80386 to create a separate virtual machine for CodeView. MagicCV uses between 4K & 8K of conventional memory as a bridge between the DOS environment and CodeView.

MagicCV is easy to use
If you are a CodeView user, you already know how to use MagicCV too. Just type MCV instead of CV; everything else is automatic.

NEW
MS Windows Version

MagicCV	$199
Soft-ICE	$386

Buy Both and Save $86!

CALL TODAY
(603) 888 - 2386
***or FAX* (603) 888 - 2465**

30 day money-back guarantee
Visa, Master Card and AmEx accepted

NU-MEGA TECHNOLOGIES

P.O. BOX 7607 • NASHUA, NH 03060-7607

MagicCV with Soft-ICE

Using Soft-ICE with CodeView gives you the features necessary for professional level systems debugging. MagicCV and Soft-ICE can work in concert with CodeView to provide the most powerful debugging platform you will find anywhere.

Both require 80386 AT compatible or IBM PS/2 Model 80. MagicCV requires at least 384K of extended memory. CodeView is a trademark of Microsoft Corporation.

Number **160** on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

The Resource Journal For IBM PC Programmers
PROGRAMMER'S Journal

Editor
M. Steven Baker
Managing Editor
Robert E. Wanetick
Contributing Editors
Michael Abrash, Frank D. Greco, Tom Swan

Contributing Writers
Michael Abrash
M. Steven Baker
Ray Duncan
William S. Hall
Rex Jaeschke
Raymond J. Michels
John Newlin
Jake Richter
Vaughn Vernon

Publisher
Elizabeth Oakley
Production
Ralph Bentley
Illustration
Jim Carpenter
Advertising
Robert Tappan
Andrea McHone
Subscriptions
Gary Hoyer
Business Manager
Bobbi Sinyard
Administrative Assistant
Lori Fanslow
Founder
Greg Estes

Business and Editorial Offices
Programmer's Journal
150 North 4th Street
Springfield, OR 97477
(503) 747-0800

Subscriptions: 1 year (6 issues) $24 ($30 in Canada and $40 elsewhere)
Change of Address
Programmer's Journal
P.O. Box 30160
Eugene, OR 97403

Articles and letters in *Programmer's Journal* represent the views of the authors and not necessarily those of the publisher. Letters to the editor become the property of *Programmer's Journal* and may be edited and published.

Code and listings in *Programmer's Journal* appear "as is," and *Programmer's Journal* makes no guarantee or warranty of their quality, performance, accuracy, or suitability for any particular purpose. *Programmer's Journal* assumes no responsibility or liability to any person or entity with respect to any damage caused or alleged to be caused by their use.

Advertising: For ad rates, deadlines, and placement, call Robert Tappan or Andrea McHone at 503-747-0800, or write to Programmer's Journal, P. O. Box 30160, Eugene, OR 97403.

Programmer's Journal is produced on Ventura Publisher, a product of Xerox Corporation.

Programmer's Journal (ISSN 0747-5861) is published bi-monthly by Oakley Publishing Company. Second-class postage paid at Springfield, OR, and additional mailing offices.

POSTMASTER: Send address changes to Programmer's Journal, P.O. Box 30160, Eugene, OR 97403.

© 1989 *OAKLEY PUBLISHING COMPANY*

from the Editor

Somehow, I still have difficulty writing these notes several months before you'll read them. As a programmer, I'm used to the immediacy of the present—getting the "#@?/" code to work. Hopefully, in time this will pass.

COMDEX for Programmers?

Fall COMDEX was bigger than ever but exhibited little new for programmers. Operating systems and networking products were well represented. Intel and Microsoft had a special exhibit on OS/2 LAN Manager with third-party vendors showing compatible network products. AT&T showed their new *Open Look* UNIX interface. IBM had a large exhibition displaying OS/2 and Presentation Manager products (OS/2 banners were all over the halls). But talking with vendors, I found that except for a handful of new OS/2 applications, most were straight ports of their DOS products.

I was most excited by the *mini-Case* tools designed for programmer productivity. I'm not a Case believer yet, at least not the full-blown Case development environments. When I visit a local IBM mainframe shop and see 40 COBOL programmers rewriting a billing system, maybe I can be convinced. But for many programming projects, I suspect (as a dumb programmer) that the preferred approach is to assemble the best package of development tools possible. Enter, the smaller *Case-like* tools that are becoming available. At COMDEX, Sterling Castle showed an interesting product called LogicGem—a logic compiler. LogicGem was designed to automate checking logic predicates and could generate source in C, BASIC, FORTRAN, Pascal, and dBASE. The user interface needed a little attention but otherwise was an interesting product. HavenTree Software was showing a prerelease 6.0 version of EasyFlow, their graphic flow-charting program. The new version available in May has added enhancements for programmers to create Data Flow diagrams and larger, more complex program structure charts. Another product worth looking at.

Strategically just before Comdex, *PC Magazine* announced its annual awards. Several programming products were deservedly selected as winners: Turbo Debugger from Borland; Lahey FORTRAN 286 and 386; and VM/386 from IGC Corporation. We add our accolades to these excellent products. One common element in all of these winners was their full-featured use of the 80386.

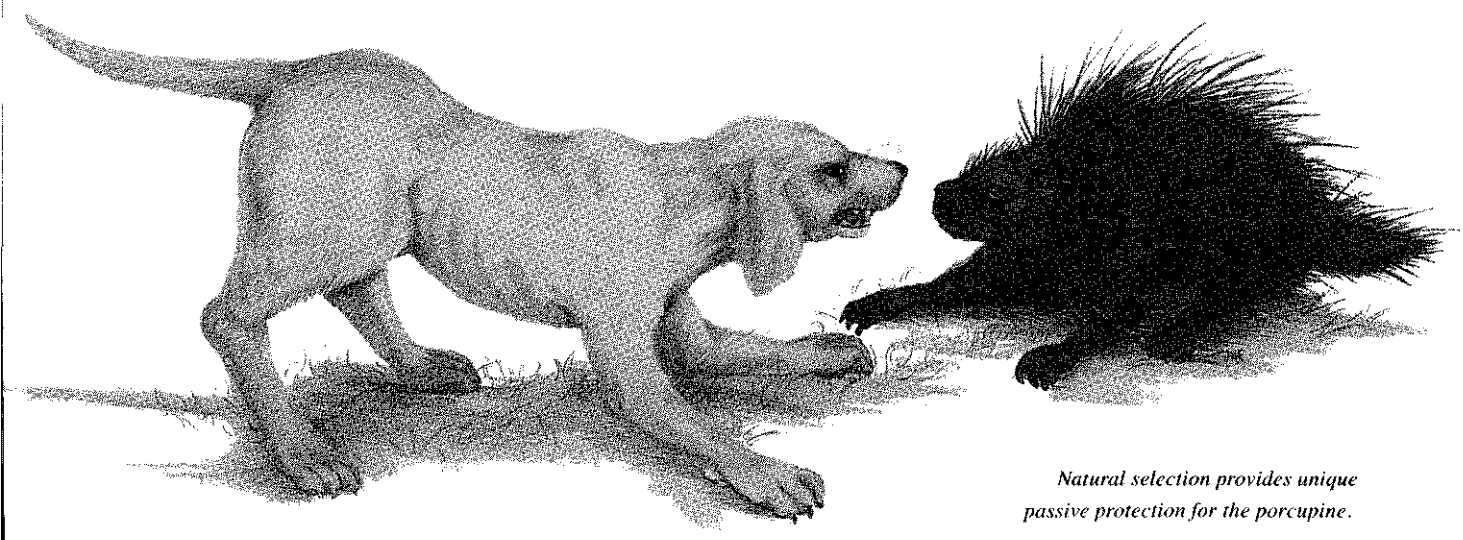
Lest We Operate in the Protected Mode!

Let's face facts, most 286 and 386 AT-class machines are still running MS-DOS. Those PC machines being used with OS/2 or 386 UNIX/XENIX are mostly new machines. Often these are developer machines, most commonly Compaqs and IBMs. This is fortunate, since potential hardware compatibility problems loom ahead in the protected mode.

The Intel 80286 CPU went through several steps (revisions) before a clean chip was generated. Most of these steps dealt with software and hardware bugs when operating in the protected mode. But under DOS, these differences were hidden from view. The 80386 has its own family of problems. Until last fall, 80386 chips were step B1 or earlier. The step D chips started appearing in machines with the availability of 25MHz parts. The B1 chips contain a number of significant bugs, most of which occur only when operating in the protected mode. A problem that doesn't exist on a December 1988 machine can occur on an older PC.

Another major concern is with machine and hardware differences. New 80386 machines sport a raft of special features (shadow RAM and caches) that differ from vendor to vendor. To fully utilize these features (some of which may actually cause problems), the operating system will need to be customized. Say goodbye to those "good old days" when the same IBM PC-DOS 3.X worked equally well on all PC machines. Disk controllers pose the most immediate problem for the protected mode, but any I/O cards or devices not fully hardware and timing compatible are potential risks. Operating under DOS, a disk controller only had to have its own ROM BIOS to be fully functional. In the protected mode, operating systems drive the I/O hardware directly for performance reasons. For evidence of this problem, talk with UNIX 386 folks about compatible disk controllers. Get away from your basic MFM controller, and compatibility problems arise. All SCSI controllers require custom drivers, and the same problems can arise with RLL and ESDI. Is OS/2 any different? Unfortunately, no! Both old and new hardware may require special device drivers. Well, who is going to write OS/2 and UNIX/XENIX device drivers for a two-year-old disk controller no longer in production? How about all those early EGA and VGA cards that don't *quite* work right? —*Steve*

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



Natural selection provides unique passive protection for the porcupine.

The Activator - Natural Selection For Software Protection

Inventor and entrepreneur Dick Erett explains how "The Activator" provides sane protection for your intellectual property.

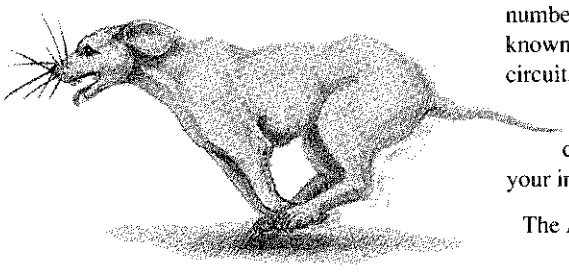
"In any industry, just as in nature, the process of natural selection raises one solution above another. Natural selection is the most elegant of engineers.

In the area of software protection The Block has been selected by the marketplace as the solution that works. Over 500,000 packages are protected by our device.

For the past 4 years our philosophy has been; *'You have the right and obligation to protect your intellectual property.'*

A New Ethic For Software Protection

In allowing end-users unlimited copies of a software package and uninhibited hard disk and LAN operation, The Block has created a new ethic for software protection.



By removing protection from the magnetic media we remove the constraints that have plagued legitimate users.

They simply attach our key to the parallel port and forget it. It is totally transparent, but the software will not run without it.

A New Technology For Software Protection

Our newest model, The Activator, builds on our current patented design, and establishes an unprecedented class of software protection.

We have migrated and enhanced the circuitry of The Block to an ASIC (Application-Specific Integrated Circuit) imbedded in The Activator.

This greatly improves speed and performance, while reducing overall size. Data protection can also be provided.

Programmable Option

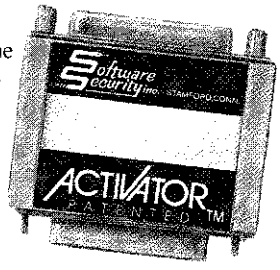
The Activator allows the software **developer** the option to program serial numbers, versions, or other pertinent data known only to the developer, into the circuit, and access it from the program.

Once you program your part of the chip, even we have no way to access your information.

The ASIC makes emulation of the device virtually impossible. It also presents an astronomical number of access combinations.

Full 100% Disclosure

Since The Activator is protected by our patent we fully disclose how it works. Once you understand it, endless methods of protection become evident.



Just as no two snowflakes are the same, no two implementations of The Activator are identical. And like the snowflake the simplicity of The Activator is its greatest beauty.

We never cramp your programming style or ingenuity. Make it as simple or complicated as you desire.

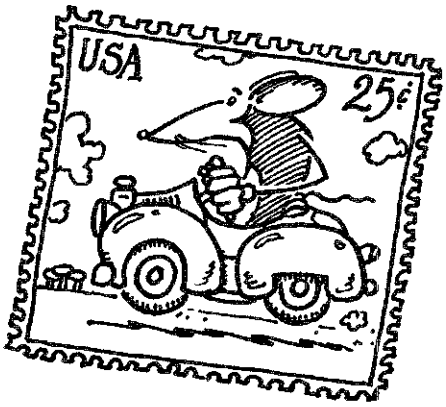
Let us help safeguard what's rightfully yours. Please call today for additional information or a demo unit. *It's only natural to protect your software.*"

1-800-333-0407 ext.103
In Connecticut 203-329-8870
Fax 203-329-7428

Software Security inc.
870 High Ridge Road
Stamford, CT 06905

Unlimited Copies • Programmable • No Batteries • Small Size • Fast • Patented • Data Protection

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



Git Along, Little Mousie

I began reading volume 6.6 with the letter from Michael Burton of Keytronic Corporation regarding the mouse update. In turn this prompted me to go back to volume 6.4 for the Pascal stuff on the mouse by John Newlin. Although it has been many years since I programmed in Pascal (at that, UCSD version), I perceived errors in his code regarding button releases and presses.

The button to investigate is correctly brought into the functions (procedures) in the BX register that utilize Mouse functions 5 or 6. However, the button values should be 0 for the left, 1 for the right, and 3 for the center (if applicable). (I have successfully used the values 0x02, 0x03, 0x04, and 0x05 for the center button. Nonetheless, I have settled on 0x03 because I generally have defined the center button as 0x04. Therefore, in continuous loop situations, I can *feed* the function the defined value, making certain I decrement the button value by one 1 inside the function. The enclosed code should make my intent clear.) Furthermore, the number of presses or releases are returned in the BX register, not the AX register!

I have recently been programming front-end interfaces using the mouse and have become somewhat embroiled with button release information. I commonly set flags true or false dependent upon release information I obtain within continuous mouse polling loops. From this I avoid repetitive button (key input) response in the cases where the user continually holds down any one button. I merely do not allow them by simply setting a release flag false and subsequently setting it true when the button is released (see Mouse function 6, INT 33H).

I am mildly surprised that Mr. Burton did not comment on this mistake. Nevertheless, I wish to thank him for the undocumented functions, particularly function 36. This is a great mouse driver diagnostic tool.

Ron Schroeder
Santa Ana, Calif.

Code from Schroeder letter

```
/*
 * mse_demo.c Ron Schroeder 11/25/88 Revised: to use ANSI escape sequences
 * to clear screen and position cursor for ease of readability
 * Purpose: to track and record number of mouse button presses and
 * releases. Must keep "running" totals because functions 5 and 6 reset
 * count to zero each time it is called - note that button presses and
 * releases returned in the register bx! not ax as seen in other code.
 * Register ax returns the status of any button being held down
 * unreleased) regardless of any value initially sent into bx.
 * Left = 01b (01h), Ctr = 100b (04h), Right = 10b (02h). Conveniently
 * these are the same as the requirements for function 3 as well as the
 * commonly defined values
 * Equipment: Logitech Three Button Bus Mouse on IRQ 3 (no com2 port)
 * Compiled/Linked using Microsoft C v5.1 (small memory model slibce.lib)
 */

#include    <stdio.h>
#include    <conio.h>
#include    <dos.h>                         /* necessary header files */

#define     TRUE    1
#define     FALSE   0
#define     LEFT    1
#define     CNTR    4
#define     RIGHT   2
        /* the following are ansi escape sequences to use,
         * make sure device=ansi.sys is in your config.sys file */
#define     CLSCR   puts("\x1B[2J")         /* clear the screen */
#define     HOME    puts("\x1B[0;0H")       /* cursor at row 0, col 0 */
#define     LINE_5  puts("\x1B[5;0H")       /* cursor at row 5, col 0 */

union   REGS    inregs, outregs;            /* for dos functions */

main()
{
        void    get_press_rel(int,int*,int*); /* function prototype */
        int     verify_mouse(void);         /* function prototype */
        int     lft_but_press = 0           /* initialize all "running */
        int     ctr_but_press = 0;          /* total" variables to zero */
        int     rt_but_press = 0;           /* in the beginning */
        int     left_but_rel = 0;
        int     ctr_but_rel = 0;
        int     rt_but_rel = 0;
        int     unk_press, unk_rel;         /* either 0 or 1 returned
                                               * from get_ function */
        CLSCR;                              /* clear the screen */
        HOME;                               /* position cursor at home */
        if (!verify_mouse())                /* init mouse to default */
        {                                   /* parameters and see if */
            puts ("Error: Mouse Not Active...");   /* it is active */
            exit (-1);                      /* quit, errorcode -1 */
        }

        do                                  /* do until any key struck */
        {
            get_press_rel(LEFT,&unk_press,&unk_rel);
            lft_but_press += unk_press;     /* "running totals" */
            left_but_rel += unk_rel;
            printf("Left Button Status:   Pressed %2d - Released %2d\n",
                    lft_but_press, left_but_rel);

               get_press_rel(CNTR,&unk_press,&unk_rel);
            ctr_but_press += unk_press;     /* "running totals" */
            ctr_but_rel += unk_rel;
            printf("Center Button Status: Pressed %2d - Released %2d\n",
                    ctr_but_press, ctr_but_rel);

            get_press_rel(RIGHT,&unk_press,&unk_rel);
            rt_but_press += unk_press;      /* "running totals" */
            rt_but_rel += unk_rel;
            printf("Right Button Status:  Pressed %2d - Released %2d",
                    rt_but_press, rt_but_rel);
            HOME;                           /* cursor back to top left */
        } while (!kbhit());                 /* until any key struck */
        LINE_5;                             /* position at row 5 after */
}                                           /* termination */

int     verify_mouse(void)                  /* function to init mouse */
{                                           /* and verify that it is */
        inregs.x.ax = 0x00;                 /* active - number of mouse */
        int86 (0x33,&inregs,&outregs);      /* buttons returned in BX */
        if (!outregs.x.ax)                  /* here I am not interested */
            return(FALSE);                  /* not active! */
        else
            return(TRUE);                   /* active, all is OK */
}

void    get_press_rel(button,press,release)   /* function to obtain nmbr */
int     button;                             /* of button press/release */
int     *press, *release;                   /* please note that button */
{                                           /* = one less than defined */
        inregs.x.ax = 0x05;                 /* value above */
        inregs.x.bx = button - 1);
        int86 (0x33,&inregs,&outregs);
        *press = outregs.x.bx;              /* # key presses per button */

        inregs.x.ax = 0x06;
        inregs.x.bx = (button - 1);
        int86(0x33,&inregs,&outregs);
        *release = outregs.x.bx;            /* # key release per button */

}
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

We Interrupt This Program . . .

I recently purchased a copy of your magazine (Vol. 6.6) to see if I would like to subscribe, but I ran into some problems with one of the programs printed in it. I am referring to Jake Richter's article, "64K Extra Memory for DOS Applications."

I typed the program exactly as it appears and then assembled it with Borland's TASM and TLINK. The first problem developed because the author failed to add to the END statement the label for the entry point. I found no trouble assembling when I changed the last line to read as follows:

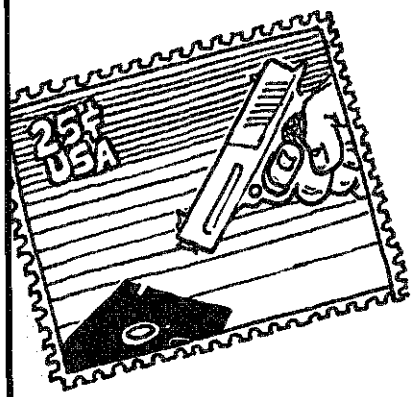
```
end         Gate20
```

After the assembly, I found that the program ran but did nothing. I have a Dell computer with a 286 processor and 1024K of RAM. I read the article again, and it seems that this program should work on my machine.

I next decided to debug the program by running it through Borland's Turbo-Debugger. As I stepped my way through the program, it hung my machine at the line that reads

```
out COM8042, AL
```

As soon as I executed that line of code, everything stopped, and I had to use the big red reset (off switch). I then tried to debug it with debug.com and got the same results at the same line. I now have a program that does not appear to work and can't be debugged! This is hot stuff. I wonder if anyone else has had this problem.



The author states in the text portion of the article that INT 15H in the BIOS modifies the state of the A20 gate, but nowhere in his program does he call INT 15H, and a review of the Norton Guides for Assembly Language failed to reveal any reference to any INT 15H function that does what Mr. Richter describes. What is the function number, and why is there no call to INT 15H in his program?

I have enjoyed the magazine but would feel better about subscribing if I felt the programs in it worked.

Howard Rensin
Silver Spring, Md.

Richter Responds

The "end Gate 20" fix that Mr. Rensin suggests is absolutely correct. MASM 4.0 does not trap that sort of omission, but it would appear that TASM does. The COM file generated by the MASM-LINK-EXE2BIN sequence still runs, though, because MASM fills the additional 100H bytes at the beginning of the program with zeros, which is the instruction "ADD [BX+SI], AL". When a COM program starts up, SI, BX, and AX are usually set to zero, so no action occurs. But it's not safe to count on those conditions. My apologies for the omission.

Secondly, running the program will not provide you with any visual effect, nor will DOS recognize the memory when you do a CHKDSK. All it does is enable the access to the 65520 bytes at linear address 10000H. This program was only meant to demonstrate the technique used in enabling/disabling access to that extra memory so that a programmer could use it in an application if he or she so desired.

A good way to test whether the program is working is to go into DEBUG and type the following lines (assuming that ENABA20.COM is in the current directory):

```
N ENABA20.COM
L
D FFFF:10
D 0:0
G
D FFFF:10
D 0:0
```

The first two dumps of data should be identical except for the addresses that appear on the left. This exhibits the "wrap" behavior I discussed in my article. After running the program (using the "G"), you should see a difference in the two dumps this time, be-

The Buck Never Stops

Because of a printer's error (No, really!), Jake Richter's article in the January/February issue (Volume 7.1) had an ad where a column of text should have been. The printer sent a corrected copy of page 38 to all our subscribers who received that issue. For the convenience of the rest of our readers, we are reprinting the missing text and table on page 92 of this issue.

—REW

THE PROGRAMMER'S CHOICE

OPTASM
- fastest assembler available
- eliminates phase errors
- never inserts spurious NOPs
- complete TSR online help
- includes a linker and debugger
- "OPTASM is clearly technically superior to MASM" *Computer Language - Dec 88*

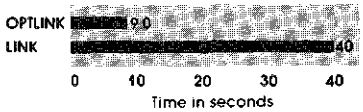
$125

ASSEMBLING BGIDEMO ON PC

OPTLINK
- fastest linker available
- generates EXE, COM or SYS
- generates global XREF
- supports CodeView info
- works with MS C5.1

$125

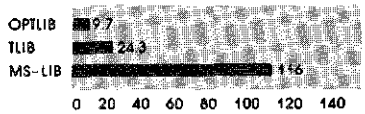
LINKING BRIEF



OPTLIB
- 10 times faster than LIB
- * and ? in filenames
- time-based replace
- LIB-compatible syntax
- rename PUBLICs and EXTRNs

$49

BUILD 670K .LIB FILE



OPTDEBUG
- source level - supports CV info
- online help system
- reverse execution
- 50k COM file
- debug EXEs, COMs and TSRs

$125

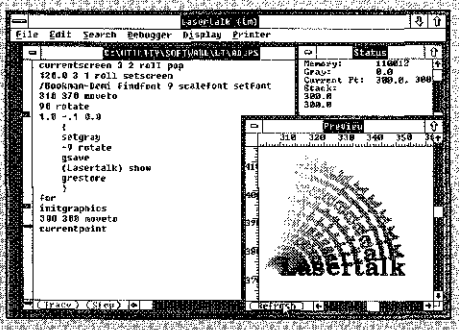
SLR SYSTEMS
1622 N. Main Street Butler, PA 16001
412-282-0864 TELEX 559215
800-833-3061 (USA)
Call today to order!

Number **165** on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Master PostScript®

Language Programming - On Your PC



Lasertalk™ PC

A PostScript language development environment which provides interactive access to the PostScript interpreter, a PostScript language debugger/editor, on-screen previewer, dictionary browser and Adobe Systems' operator descriptions, online. Requires PC/AT or greater and a PostScript printer. Includes Microsoft® Windows runtime. Suggested Retail Price $399. For more information, see your dealer or contact Emerald City Software at (415) 324-8080.

EMERALD CITY SOFTWARE

800 Menlo Ave, Suite 102, Menlo Park, CA 94025
(415) 324-8080

Lasertalk is a trademark of Emerald City Software. PostScript is a registered trademark of Adobe Systems, Inc. Microsoft is a trademark of Microsoft Corporation.

Number **241** on the Reader Service Card

cause the A20 gate is enabled. The last "D FFFF:10" is now displaying data from the memory at the one-megabyte boundary.

Next, I probably should have stated that you can't single-step through the code with a debugger, since it plays with the 8042 Keyboard Controller. The commands sent to the 8042 effectively disable other types of keyboard operations, such as keystrokes. This is why the code starts by disabling interrupts and then reenables them after it has completed its operation. This also explains why Mr. Rensin had to cold-boot his system: His keyboard was disabled by the "OUT COM8042, AL" instruction.

Lastly, I mentioned INT 15H in the article because certain INT 15H calls (specifically the ones that access extended memory) change the state of the A20 Gate, usually closing it. An INT 15H call is not made in the code I supplied because INT 15H does not give the calling program control over the A20 Gate state. INT 15H just happens to be one of the known pieces of code that mucks with the A20 Gate.

I hope this clears up any confusion that may have arisen from my article.

Jake Richter
Woburn, Mass.

TurboMagic Solutions

I found Tom Swan's article "Demagnetizing Static Variables" in *PJ* 7.1 interesting. His solution to saving modified values of typed constants to an EXE file is similar to the solution that we developed for the installation utilities that we include in our TurboMAGIC package. There are a couple of differences worth mentioning.

Both solutions use special **begin** and **end** markers (**CBase** and **EBase**, respectively, in Tom's solution) to delimit the installable typed constants. To locate the installable typed constants in an EXE file, Tom does a forward search of the EXE file for the **begin** marker.

In contrast, the TurboMAGIC utility does a backward search of the EXE file for the **end** marker. Since the typed constants are near the end of the file, this significantly

PJ CODE

Programmer's Journal Code DiskScription—

Code disks from six issues, only $39.95*

Typing in long code listings uses up a lot of valuable time, and calling up bulletin boards doesn't appeal to everyone. So what does the modern programmer do? Get a ***PJ Code DiskScription***, of course!

What's a PJ Code DiskScription?
All the code listings from six consecutive issues of *Programmer's Journal*, sent directly to you every other month on disk, for the money-saving subscription price of only $39.95.*

You'll get all the upcoming installments of Michael Abrash's EGA/VGA ASM code. You'll get more of Tom Swan's handy Pascal code. You'll get great code from all ***PJ***'s authors, some of the country's hottest programmers.

You'll receive your code disks in the mail every other month throughout the year. Learn from ***PJ*** code or use in it your own programs. Each disk contains all of the program listings published in ***PJ*** plus .EXE files, .OBJ files, benchmark source files, all for **only $39.95!***

If purchased separately, the regular price for these six disks is $60.00. **So save yourself some time, effort and over $20.00—order your *PJ DiskScription* today!**

*GUARANTEE: If you are dissatisfied with your **PJ Code DiskScription** for any reason, we will gladly send you a full refund on all unmailed diskettes.*

PJ CODE DELIVERED DIRECTLY TO YOU—
GET YOUR *PJ Code DiskScription* TODAY!

All *PJ Code DiskScriptions* must be prepaid.
Return the card on the opposite page with your payment.
For faster service with VISA MC call toll-free: 800-234-0386

* **Prepaid, US only.** Call or write for prices outside the US.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

HARDCORE SOFTWARE

Stop fooling around. It's time to get hardcore about software. With Microsoft.

We'll give you all the resources you want. Tens of millions in R&D funding. Along with one of the most elementary tools for thinking — a door, which leads to your own private office. All backed by management that truly *does* speak your language, because they probably helped write it.

We're serious about software design. If you are too, then apply right now for one of these opportunities.

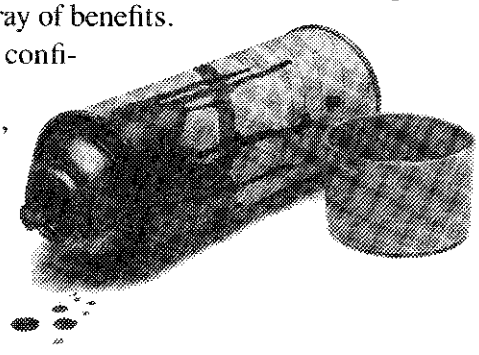
Software Design Engineers

We're working on everything from worldclass application software, compilers, operating systems, and networking to sophisticated graphics, powerful productivity software and more. You could be too, if you have programming experience and a background that includes 68000, Macintosh® Toolbox, Windows™, "C", micro's, 8086, UNIX™/XENIX®, or MS-DOS®.

There are opportunities to work with our teams in systems, applications, CD-ROM, languages, or networking.

If you qualify, Microsoft offers you an opportunity to live and work where the quality of life is high and the cost of living is low — the beautiful Pacific Northwest. Along with amenities such as a health club membership, workout facilities and parcourse, plus an array of benefits.

To apply, please send your resume in confidence to: MICROSOFT CORPORATION, Human Resources, Dept. CWSDE-0389PJ, 16011 N.E. 36th Way, Box 97017, Redmond, WA 98073-9717. We are an equal opportunity employer. No phone calls, please.



Microsoft®

Trademarks are registered to their respective companies.

Number **232** on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

reduces the search time. And it virtually eliminates the need for Tom's performance-enhancement technique of running the installation program once to determine the location of the **begin** marker and then hardwiring the location into the source code for the installation program.

Both solutions use the difference in the address offsets of the two markers in memory as the number of bytes to save to the EXE file after the user has completed installation. This is the only use that Tom makes of the difference in locations.

The TurboMAGIC utility goes a step further. It compares the difference in the address offsets of the two markers in memory with the difference in the marker locations in the EXE file. If they're not the same, the program aborts. This check, while not foolproof, does provide some assurance that the installable typed constants in the EXE file and those in the installation program match.

Dalton Orr
Sophisticated Software
Mobile, Ala.

Saving EGA/VGA Screens

Here is an alternative to Michael Abrash's method for saving EGA/VGA screens ("Saving Screens, Color Selection, and Other Mysteries," *PJ* 6.6). Michael provides a method in which each video plane is stored to disk and then later restored. While this works just fine, the file size is 110KB for an EGA screen and 150KB for a 640x480, 16-color VGA screen. For the new, extended 256-color modes, a 640x480, 256-color screen takes over 300KB!

My method involves "cheating" a little and using a package called the PCX Programmer's Toolkit. With the toolkit, any CGA, EGA, VGA, or extended VGA screen can be saved and redisplayed. The steps to save an EGA screen in C would be

```
pcxSetDisplay(pcxEGA-10);
pcxSetMode(pcxGRAPHICS);
pcxDisplayFile("MyIMAGE.PCX",0,0,0);
```

The first two statements are simply initialization statements—the third function actually does the screen saving. The three 0 parameters are used for x, y, and page location. To redisplay the image file, just use

```
pcxFileDisplay("MyImage.PCX",0,0,0);
```

The advantages of these functions are manyfold. The screens are saved in the PCX format, which is a compressed format originally developed by ZSoft Corporation for PC Paintbrush. The PCX format is a de facto standard for bit-mapped graphics and is supported by most desktop publishing software, including PageMaker and Ventura. Thus, all saved screens not only are compressed but also can be loaded by PC Paintbrush, PageMaker, and Ventura and by demo packages like Proteus (by Helios). Screens are saved and displayed on a line-per-line basis instead of plane by plane; so, images appear smoothly. To save a CGA, VGA, or Hercules screen, simply set a different display type—the code remains the same. The functions also support saving only portions of the display, palette manipulation, extended VGA screens, and more.

The toolkit supports twelve different compilers and six different languages and is written entirely in assembly language. For assembler buffs, source code is available. The source code illustrates how to program the display hardware and is unique in that it supports all languages and compilers with the same source! For anyone interested in mixed-language programming (including non-Microsoft products), the source includes a set of macros that handle everything automatically.

Christopher A. Howard
Genus Microprogramming
Houston, Texas ♦

SCREEN MANAGER
P·R·O·F·E·S·S·I·O·N·A·L

ADVANCED WINDOWING

• The Screen Manager Professional gives your applications the finest bullet proof windowing interface available. All functions have been carefully designed and tested with the programmer in mind. The functions are easy to use and remember. Although other products may boast of hundreds of functions, they often involve needless repetition. Our functions are elegant and powerful. Each accomplishes a variety of tasks with one command. All functions are written in hand-optimized assembly language to provide the fastest possible response and smallest possible size.

ORDER

• To receive your copy of the **Screen Manager Professional, send $139.95.** VISA, MasterCard, Check, or Money Order accepted. Sorry, no C.O.D.'s. Please allow 2-4 weeks for delivery. PA residents add 6% sales tax.

IMPORTANT FEATURES

- Unlimited number of windows
- Complete writing and scrolling support for overlapping windows
- VGA, EGA, CGA, MGA support
- Write to video ram or use BIOS
- Multiple monitor support
- No fuzz or flicker
- Drag windows around the screen
- Menu system
- Resize and reposition windows
- Save and Load windows from disk
- MS-DOS memory allocation
- 100% hand-optimized assembly
- Over 90 powerful functions
- Royalty free!

FOR USE WITH

• Datalight C • Microsoft C
• Microsoft Quick C • Aztec C • Turbo C
• Microsoft Quickbasic • Turbo Pascal

EASY TO USE

• Free yourself to concentrate on the substance of your program. Over 90 functions provide all the windowing tools needed to develop serious applications whether you are a professional or a hobbyist.

• The Screen Manager Professional is a powerful windowing facility. You can increase your productivity as well as the overall quality of your work. The most important part of any program is its ease of use. The Screen Manager Professional can help you create easy to use, visually stimulating, programs with a minimum of effort. Now powerful windowing is available to you at an affordable price. If you have further questions, please call or write.

Programming Utilities for Professionals

LAI
LOGICAL ALTERNATIVES INCORPORATED
Calder Square
P.O. Box 10674
State College, PA 16805-0674
(814) 234-8088

Number **246** on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Reselling

by
Jake Richter

Reselling computer products does not just mean getting rid of your old equipment. Rather, reselling is a legitimate business involving ordering items directly from distributors and manufacturers—generally at discounts of 20 to 60 percent off retail, depending on volume—and selling them to end-users. One benefit of being a reseller is being able to find bargains in new hardware and software for your own development needs without having to depend on mail-order houses, overstocks at local computer stores, or friends.

A computer-products reseller can be one of several types of entities:

- *Computer dealer.* This type of reseller specializes in selling computer equipment to end-users at retail prices.
- *Original-equipment manufacturer (OEM).* An OEM takes raw computer parts, such as drives and motherboards, and incorporates them into saleable products with his or her company's label on the outside.
- *Value-added reseller (VAR).* A VAR is similar to an OEM, but the VAR takes a finished product and adds some value to it, usually via some vertical market software, before reselling it. A VAR can also be considered a systems integrator.
- *Consultant.* This is the most generic form of reseller, one that virtually anyone reading this magazine can call himself or herself. Many consultants offer services similar to that of VARs and even OEMs. They set up systems, install software, sell the systems, and train the purchaser. To do this profitably, the consultant must be able to purchase equipment and software at prices below retail, and this necessitates becoming a reseller.

You only need one thing to qualify as a reseller: a resale certificate, which you can obtain most likely from your state Department of Revenue (DOR). In Massachusetts, a resale certificate costs $10. Your resale number will usually be the same as your federal tax identification number, which is usually the social security number for an individual or DBA (see sidebar) and the employer identification number (EIN) for a corporation.

DBA

DBA stands for "doing business as." For example, if your name is Fred Smith, you might want to form a company called Fred Smith Associates, or Fred Smith & Co., or SmithSonics. The company name is the DBA, which you must legally register.

The procedure for registering a DBA varies from state to state. In New York, for example, you can get a DBA form from a stationery store, fill it out, and go to the town hall to file it. In New Hampshire, you have to register with the state. In Massachusetts, you should talk to someone at your town hall (in Woburn, it turned out to be the building inspector). A DBA will usually cost $10 to $40.

A DBA is the way to obtain a name for a company without the legal help that partnerships and corporations require. On the other hand, you don't get the legal protection from lawsuits that incorporation provides. Some towns may not allow you to register a DBA at a residentially zoned address, even if you tell them all your business is conducted by phone (that is, you have no visiting clients). Some states and towns require that you also post an official notice of DBA in the local paper. Contact an attorney or your local authorities for local requirements.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

As a reseller, you are assumed to be purchasing goods to sell to others. In most states, sales of goods are taxed by a mechanism known as "use tax," or sales tax. In Massachusetts, resellers are required to turn in collected sales tax quarterly or monthly, depending on volume. Failure to charge sales tax to Massachusetts residents can be cause for severe penalties. To find out how your state handles use tax, call your state revenue office.

Once you have your resale certificate and number in hand, you might find it useful to have business cards and letterhead stationery with your company's name on them. It is not always necessary, but it will help present you as a professional when you deal with distributors and manufacturers.

Ordering from Distributors

Distributors generally are companies that sell products to dealers at wholesale prices. Some smaller distributors will also sell products to end-users, but only at full retail price. Distributors may focus on vertical markets, such as graphics, or handle a large variety of different products. With larger distributors, you should expect somewhat better prices and service, but they may not carry some of the more specialized and scarce items.

The best way to start your reselling business is to contact the big national computer distributors, such as Softsel, Microamerica, Tech Data, and Micro D (see box on page 16). I have found Softsel and Microamerica the easiest to deal with. When you contact a distributor, identify yourself as a consultant and computer reseller, and ask them to send you their latest catalog and price list. Be sure to have your resale number handy, in case they ask for it.

Most of these distributors will put you on the regular customer list only after you purchase a product from them. Most distributors will also have you fill out a credit application to determine on what terms they should carry you. There are a variety of terms for which you may qualify:

- *Prepayment only.* If the distributor doesn't feel comfortable with your (or your company's) financial history, they will request that you prepay all orders via mail or wire. This is a rather inconvenient way of doing business, but if it's all you can get, it's still better than paying full price. Some of these companies will allow credit card payment, which is far more convenient than prepaying with cash.
- *COD—cash or cashier's check.* This method is much more common than prepayment and allows distributors to feel confident that they are receiving funds that won't bounce. Money orders are another form of cashier's check, but they generally can be obtained for amounts less than $1,000, are cheaper than cashier's checks, and can be purchased at most major supermarkets as well as banks.

  The disadvantage of this form of payment is that you need to have the exact amount available when the delivery arrives. Also, shipping companies are notorious for showing up when you're not around.
- *COD—company check.* This is the most common form of payment besides credit, which is harder to qualify for. The requirement here is that you have company checks. (If you are not doing business as a company, make sure to be on personal-check terms with the distributor.) With this status, you are given a dol-

PROTECT INTELLIGENCE WITH INTELLIGENCE

PROTECH KEYs:

- ❑ are totally transparent to the end user
- ❑ allow unlimited back up copies
- ❑ free up disk drives
- ❑ do not interfere with peripheral operations
- ❑ are easy to install
- ❑ protect software developers' revenues
- ❑ provide the highest level of protection available

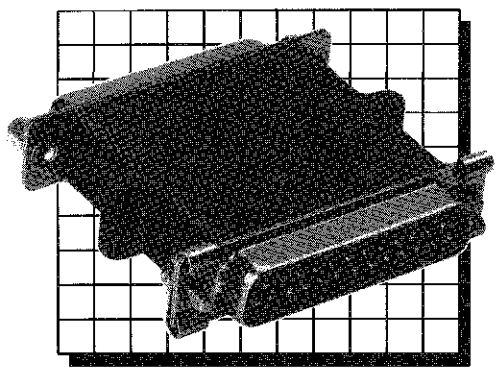
PROTECH'S KEYs provide the highest level of protection because of:

- ❑ assembler based routines (not drivers)
- ❑ encrypted routines
- ❑ physically unique hardware KEYs
- ❑ sophisticated software installation advice

THE KEY

The KEY provides effective software protection while insuring customer satisfaction. The KEY is a random response device which is designed for identically reproduced software packages.

SOFTWARE PROTECTION KEYs BY PROTECH



WE HAVE THE *KEY* TO SOFTWARE COPY PROTECTION

For a demonstration package or additional information, please write or call:

1-800-843-0413

THE MEMORY KEY

The MEMORY KEY is a programmable software protection device. Each byte of memory can be addressed in groups or individually. Possible applications for the MEMORY KEY include:

- ❑ modular package control
- ❑ serialization
- ❑ customization
- ❑ access control to PCs

Information stored in the MEMORY KEY can also be altered by your application software in the field. This dynamic writing to the MEMORY KEY opens up unlimited possibilities. Possible applications for the dynamic read/write function include:

- ❑ demo control
- ❑ software leasing
- ❑ updating modules in the field
- ❑ any "counter" operation

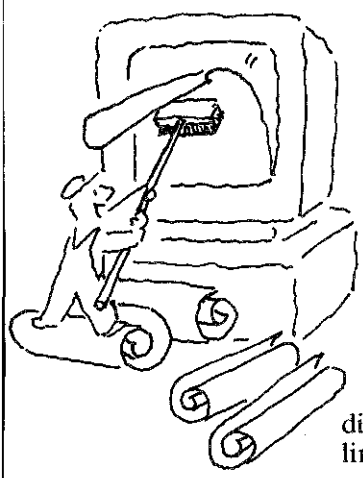
PROTECH MARKETING, INC.
1804 W. Southern Parkway • Building A-112
Durham, North Carolina 27707
(919) 490-4970 FAX (919) 490-4974

Number 228 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Two great tools.

SAYWHAT?!
The lightning-fast screen generator.



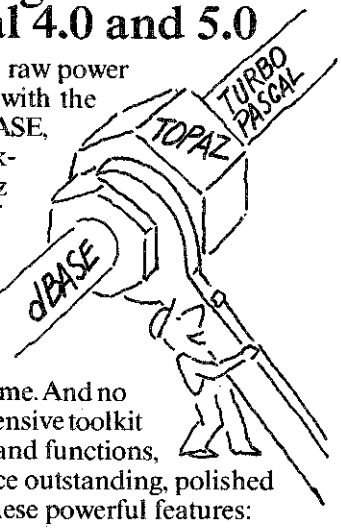
Whether you're a novice programmer longing for simplicity, or a seasoned pro searching for higher productivity, you owe it to yourself to check out Saywhat. You see, with Saywhat, you can build beautiful, elaborate, color-coded screens in minutes! That's right. Truly *fantastic* screens for menus, data entry, data display, and help-panels that can be displayed with as little as one line of code in *any* language.

Here's what you get:

- Design screens, windows, and *moving bar menus!*
- Easy-to-use, powerful editor lets you create screens in a jiffy.
- Pop up your screens and menus with one line of code in dBASE, all the dBASE compilers, your favorite BASIC, Pascal, *or any other language!*
- Screen Library Manager.
- Generates runtime code.
- No runtime license or royalty fees.
- Comes with a 100 page manual, plus dozens of sample programs and free utilities.

$49.95

TOPAZ
The breakthrough toolkit for Turbo Pascal 4.0 and 5.0



If you'd like to combine the raw power and speed of Turbo Pascal with the simplicity and elegance of dBASE, Topaz is just what you're looking for. That's because Topaz (our brand new collection of units for Turbo Pascal 4.0 and 5.0) was specially created to let you enjoy the best of *both* worlds. The result? You can create truly dazzling applications in a very short time. And no wonder. Topaz is a comprehensive toolkit of dBASE-like commands and functions, designed to help you produce outstanding, polished programs, fast. Check out these powerful features:

- Over 200 routines all with easy-to-use, dBASE-like syntax.
- Data entry routines like **SAY, GET, PICTURE, RANGE,** color selection, unlimited data validation.
- Open up to 10 DBF files, with up to 7 indexes each with database routines like **USE, SELECT, SKIP, APPEND, PACK, INDEX ON, SET INDEX TO,** and **FIND.**
- No need to buy dBASE. **CREATE, BROWSE** and **REPORT** utilities included.
- Easily implement Saywhat and Lotus-style moving bar menus.
- **BROWSE** any DBF file with just one line of code! Programmable and windowed too.
- Comprehensive Time & Date math in 7 international formats.
- Powerful *code* and *REPORT generators* included!
- Comes with a complete 250 page manual, plus sample programs to get you started.

$49.95
Special introductory price! (Until March 31, -then $74.95)

ORDER NOW. YOU RISK NOTHING.

Thousands of satisfied customers have already ordered from us. Why not visit your dealer or call toll-free, right now and put Saywhat and Topaz to the test yourself? They're fully guaranteed. You don't risk a penny.

Special limited-time offer! Save $15. Buy Saywhat?! and Topaz together before March 31, 1989 for just **$85** (plus $5 shipping & handling).

Visit your nearest dealer or call toll-free:

800-468-9273

In California: 800-231-7849
International: 415-571-5019

The Research Group
100 Valley Drive, Brisbane, CA 94005

MONEY BACK GUARANTEE.
If you aren't completely delighted with Saywhat or Topaz, for any reason, return them within 30 days for a prompt, friendly refund.

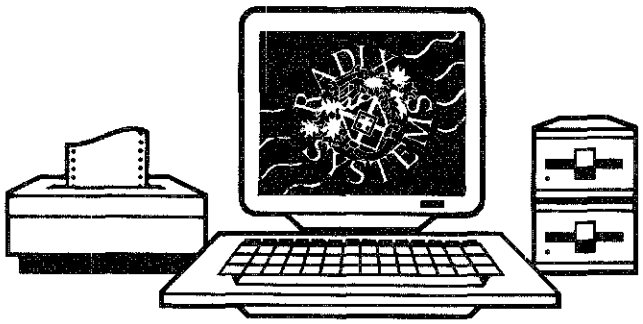
Dealers: SAYWHAT?! and TOPAZ are available from Kenfil Distribution, and in Europe from ComFood Software, W. Germany (2534-7093)

Guaranteed!

THE RESEARCH GROUP

Number 198 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



BRING DATABASE POWER TO YOUR SCREEN

Now you can achieve database flexibility in screen processing with the Radix Screen Management System. It's a language independent screen and keyboard controller. The package includes a facility that allows full screen editing of screen maps. Once created, a screen map is allocated and released at runtime, no compiling or linking necessary. Use the screen map in different applications developed under different languages. There are only eight commands to learn which control the screen maps. In addition, an Internal Document Facility will automatically display and scroll a window of your application's help text.

Other features include:

- **Window and Overlay Screens**
- **Alter Attributes at Runtime**
- **Move Data Field-by-Field or in groups**
- **Request Only Modified Values**
- **Update the Screen Map Anytime**
- **Extensive Documentation**
- **Uses Standard Command Interface**

To order, send $198 plus $7 for shipping. No royalties.

Requires an IBM PC or compatible, 512KB of memory, and DOS 2.0 or higher. UNIX and OS/2 versions will be available soon. For more information call (201) 697-2630, or send $5 for demonstration diskette and Concepts & Facilities manual.

RADIX SYSTEMS INC.
P.O. Box 306
Rochelle Park, NJ 07662

Number **251** on the Reader Service Card

PERSONAL REXX 2.0

NEW PERSONAL PROGRAMMING POWER

Personal REXX is a powerful general-purpose language bringing mainframe command and macro language capabilities to the PC. Personal REXX 2.0 is a new, faster version that runs larger programs, and features:

- **A superset of IBM CMS REXX**
- **The ability to develop powerful batch procedures**
- **Compatibility with IBM SAA REXX command language**
- **Windowing and menu support**
- **Easy prototyping**
- **The ability to interface with KEDIT™ text editor and HyperTerm™ communication package**

Personal REXX runs on the IBM PC, AT, PS/2, and true compatibles using DOS 2.0 or later. A Programmer's Journal review said: ***"Everyone who tries it seems to love it."*** Call to find out why, and how you can enjoy the programming power and the many benefits of Personal REXX 2.0

Personal REXX Version 2.0 is available at $150. Add $3 shipping. MC, VISA, American Express.

MANSFIELD
Software Group
P.O. Box 532, Storrs, CT 06268
(203) 429-8402

KEDIT is a trademark of the Mansfield Software Group, Inc.; HyperTerm is a trademark of Quercus Systems; IBM and IBM PC, AT and PS/2 are trademarks of International Business Machines.

Number **233** on the Reader Service Card

lar limit on the amount of merchandise you can have on order at any time.

You should maintain a checking account with at least $1,000 in it to qualify for COD status with the companies you plan on paying with company checks. Ideally, you should open the checking account at least six months before placing your first order. Some shippers are known for stubbornness when it comes to accepting company checks for COD shipments.

- *Credit.* Being able to receive merchandise without paying for it in advance or upon delivery usually requires a proven financial history. For someone just starting out, this may be quite difficult to establish. There are many varieties of credit and many credit plans, which vary from distributor to distributor.

One of the trouble spots common to all methods of ordering equipment is product delivery. Fortunately, some enterprising individuals spotted this as a market and came up with the concept of a "mail receiving service." Companies such as Mail Boxes Etc. make money by receiving UPS and U.S. mail packages and letters for their customers. This allows you more freedom, especially if you have to be away from a suitable delivery site during the day. You can handle CODs by leaving the payment with the receiving service in advance. Some companies also offer fax and telex services. Just look under "Mail Receiving Services" in your Yellow Pages.

Ordering Direct from the Company

Some companies don't use distributors, and some companies require that only "authorized" resellers purchase their products. The latter applies even to some products sold through distributors. One company with this restriction that comes to mind is Aldus, the company that brought us PageMaker.

If you want a certain company's product but can't find it listed in distributor catalogs, call the company directly and ask for dealer sales. Tell them that you are a reseller and are trying to obtain a copy of their product. The salesperson will then point you to a national or local distributor who sells the product, try to sell you the product directly, send you an application to become authorized, or offer you an evaluation copy at a reduced rate. Some companies,

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Make it with the best!

Speed program development time with OPUS Make.

OPUS Make tracks your executables and libraries better than any other Make utility on the market:

- DOS and OS/2 executables.
- DOS version uses **only 3K of memory!**
- Read and Run-time conditional directives.
- Loop, Include and Error directives.
- Multiple directory support.
- Extensive macro capabilities.
- DOS version supports OS/2-like standard error redirection.
- POLYTRON PVCS logfile support. Burton TLIB and MKS RCS support coming.
- UNIX makefile syntax.
- Microsoft makefile compatibility mode.
- **Also comes with OPUS MKMF:** An automatic dependency generator that fully understands C preprocessor directives.
- Comes with a 30 day money back guarantee.

OPUS Make and MKMF are writtten by the author of NDmake.

$79 + $3 S&H

OPUS Software

1032 Irving St. #439, San Francisco, CA 94122

(415) 664-7901

Number **254** on the Reader Service Card

Celebrating our 10th Anniversary!

AMX™ 86

KADAK's engineers bring years of practical real-time experience to over 600 installations world-wide.

This *real time*
MULTITASKING KERNEL
simplifies *real life*
product development

- No royalties
- IBM PC DOS® support
- C, PL/M, Pascal
- Preemptive scheduler
- Time slicing if needed
- Essential source code included
- Intertask messages
- Dynamic operations
  - task create/delete
  - task priorities
  - memory allocation
- Event Manager
- Semaphore Manager

AMX86™ v2.0 for 8086/88, 80186/88, 80286 systems

Demo package	**$25 US**
Manual only	**$75 US**
AMX86 system	**$2195 US**
Full source Add	**$805 US**

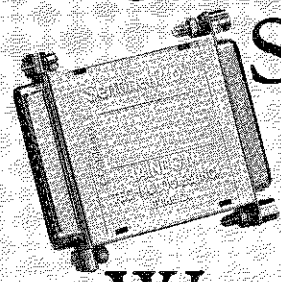
(Shipping/handling extra)

Also available for 8080, Z80, 68000

KADAK Products Ltd.
206 - 1847 W. Broadway
Vancouver, B.C. Canada V6J 1Y5
Telex: 04-55670
Fax: (604) 734-8114
Telephone: (604) 734-2796

Number **211** on the Reader Service Card

Suppose You Protected Your Software & The Silence Was Golden.



Software protected by Rainbow Technologies' Software Sentinel™ has a sound of its own: Golden. And unlike copy-protected software, very silent.

Golden because your development investment is protected. Down to each and every copy you sell. To give you the long-term solution to revenue loss. And the sure way out of the worldwide free-distribution network.

Silent because Software Sentinel protection is transparent. With easy-to-live-with reliability. And no impact on the user or your software's performance. Which means all you hear are the nods of approval.

The Software Sentinel key is shipped with the software and easily installed on the PC's printer port. Protected software will run only when it's installed. No key – No use. It's that simple.

Key Sentinel Family Features.

□ "Execution control" technique prohibits unauthorized use of software □ Permits unlimited backup copies □ Virtually unbreakable □ For DOS, OS/2 and XENIX □ Pocketsize Transportable □ Transparent operation □ Use with a hard disk

SentinelPro™

- Runs under DOS, OS/2 and Xenix on IBM PC/XT, AT, PS/2 and compatibles
- Algorithm technique (Secure, fixed response)
- Parallel and serial versions
- Minimal implementation effort
- Higher level language routines included
- 100 user-definable time-out response levels per unit
- Anti-tamper design for reliability

Sentinel-C

- Protects multiple packages with one device
- 126 bytes of user-definable memory programmed before shipment of the software
- Rainbow supplies the developer utility to write the programming the user
- Higher level language routines included
- Runs under DOS, OS/2 and XENIX on IBM PC/XT/AT, PS/2 and compatibles
- Parallel port version only
- Evaluation kit available

RAINBOW TECHNOLOGIES™

Number **222** on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

such as Microsoft, will do all of the above.

Once you are signed up as an authorized reseller, you can purchase dealer copies ("not for resale") at a much reduced price.

Uncle Sam

Once you start purchasing goods as a reseller, you should also start keeping records to protect yourself from potential tax problems. This means issuing purchase orders for each order you place and invoices for each set of items purchased from your company, even if you purchase them yourself. You may be liable for sales tax for goods that you keep for your own use. Again, check with your state revenue department to get all the details.

These tips should get you started on the road to your own reselling business—and cheaper computer products for your own use—without a great deal of overhead. Please note that I am not a lawyer; contact one for legal advice.

Jake Richter is president of Panacea, Inc., a PC consulting firm specializing in graphics processors. Readers may contact Jake on MCI Mail or BIX; his user name is JRICHTER. ♦

NATIONAL DISTRIBUTORS

Microamerica
1455 Concord Street
Framingham, MA 01701
800-343-4411 or 508-877-8500

Micro D
2801 South Yale Street
Santa Ana, CA 92704
800-642-7631

Softsel

Eastern Sales Office:
26 Law Drive
Fairfield, NJ 07006
800-645-7779 or 201-575-5245

Central Sales Office:
176 Mittel Drive
Wood Dale, IL 60191
800-645-7775 or 312-773-8444

Western Sales Office:
546 North Oak Street
Inglewood, CA 90312-6080
800-645-7778 or 213-412-1700

Tech Data Corporation
5777 Myerlake Circle
P.O. Box 6260
Clearwater, FL 33518
800-237-8931 or 813-539-7429

MARKET YOUR SOFTWARE

The seminar that has software developers talking is back. Attend TechSel's "Channel of Profit: How to develop profitable resellers in the U.S."

- March 8 - New York
- March 14 - Los Angeles
- April - Toronto

Registration $250 / Seating is limited

800•645-2287

(in NY 516 277-7000)

Ask for the Software Seminar Group

(9am - 5pm EST)

Number **264** on the Reader Service Card

C Language Training Seminars

Learn C language fundamentals.
4 days of hands-on, interactive coding sessions and lectures.
Two textbooks and over 25 source code programs provided.
Focus on Microsoft C and Turbo C.

Classes held monthly at The C-Team.
Corporate training on location performed.
Tuition: $995.

Call for information.

The C-Team, Inc.

Providing C expertise for all training and programming needs.

60 East 12th St.
New York, NY 10003
212/477-3990

Number **255** on the Reader Service Card

MEMO:

Take a break from your programming. Relax with an interactive fiction MegAdventure from MISOSYS. Adventures sharpen your wits, and keep you on your toes. MegAdventures challenge you like no other.

Pit your wits against an ill-tempered dragon. Track it down in the mysterious northern mountains, the likes of which only heart-quickening tales of terror and death are told? Lair of the Dragon contains well over 160 locations, over 200 recognizable objects, and has a vocabulary of over 600 words.

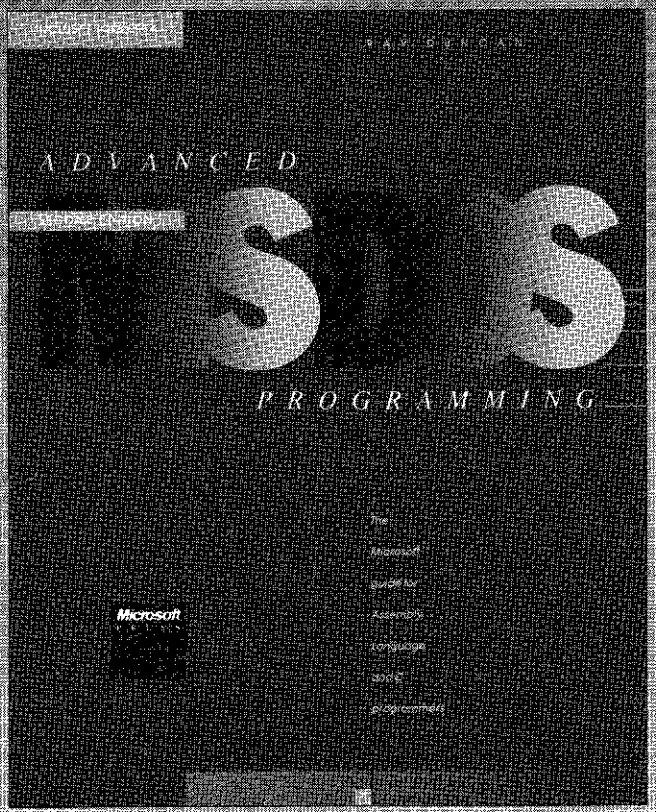
Lair of the Dragon [M-86-021]: Just $29.95
S&H: $2 US, $3 Foreign. MC & VISA.

MISOSYS, Inc.
P.O. Box 239
Sterling, VA 22170-0239
800-MISOSYS or 703-450-4181

Number **238** on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

We couldn't have said it better.



"Advanced MS-DOS PROGRAMMING exemplifies how a highly technical book can be both informative and readable.... Duncan's strengths include a style that is at once easily read, a thorough coverage of the subject matter heretofore unknown, and the frequent use of examples in the form of assembly language programs and code fragments."

BYTE magazine

"Makes good reading out of even the most elaborate technical descriptions."

Online Today

"One of the most authoritative in its field.... The book deserves a place on the shelf of everyone who has ever given a fleeting thought to programming the IBM PC and compatibles."

PC Magazine

And you can bet they'll be saying it again. ADVANCED MS-DOS PROGRAMMING—the preeminent source of MS-DOS information for assembly-language and C programmers—has just been expanded and completely updated. Included is a wealth of new data and programming advice in several significant areas:

- ROM BIOS for the IBM PC, PC/AT, PS/2, and related peripherals including disk drives, video adapters, and pointing devices
- MS-DOS through version 4
- version 4 of the Lotus/Intel/Microsoft Expanded Memory Specification
- writing "well-behaved" vs "hardware-dependent" applications
- compatibility considerations for OS/2

Ray Duncan, DOS authority and noted columnist, explores key programming topics including character devices, mass storage, memory allocation and management, and process management. In addition to his expert advice, he has packed his book with a healthy assortment of updated assembly-language and C listings that range from code fragments to complete utilities. These include a fully functional terminal-emulation program, a nifty DOS shell, and the framework for customized critical-error interrupt handlers.

And the reference section in ADVANCED MS-DOS PROGRAMMING, detailing each MS-DOS function and interrupt, is virtually a book within a book.

ADVANCED MS-DOS PROGRAMMING. Your key to fast, efficient, robust programs. **$24.95**

Microsoft® Press

Hardcore Computer Books

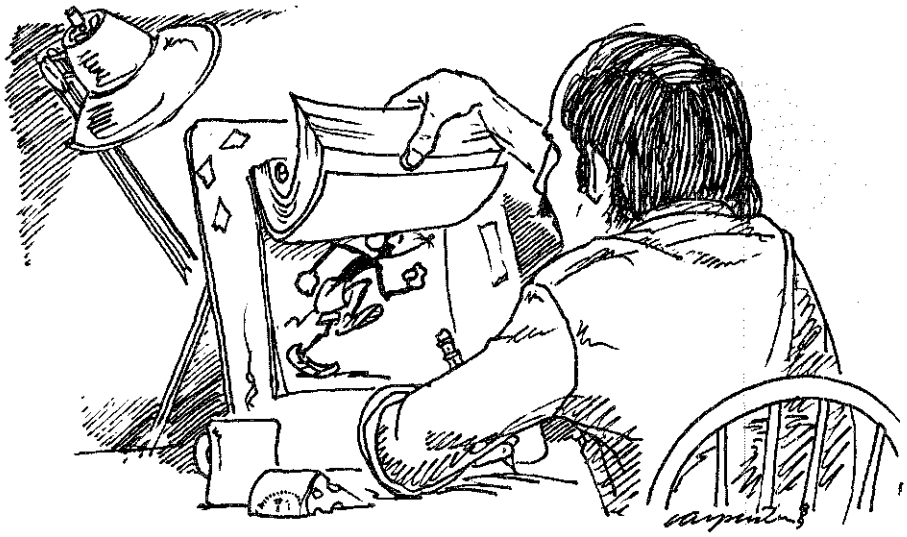
Available wherever books and software are sold including Software Etc. stores.
Or call in your credit card order ***1-800-638-3030*** *(8AM to 4PM EST).*

Number **234** on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

EGA and VGA Animation

by
Michael Abrash



Installment 11: In which we have some fun with animation, in the process refuting the claim that the PC is a lousy animation engine.

When it comes to computers, my first love is animation. There's nothing quite like the satisfaction of fooling the eye and creating a miniature reality simply by rearranging a few bytes of display memory. What makes animation particularly interesting is that it has to be performed fast (as measured in human time) and without blinking and flickering, which can destroy the illusion of motion and solidity. Those constraints make animation the toughest graphics challenge—and the most rewarding.

It pains me to hear industry pundits downgrade the PC—especially the EGA and VGA—when it comes to animation. Okay, I'll grant you that the PC isn't an Amiga and never will be, but then neither is anything else on the market. The EGA and VGA offer good resolution and color, and while their hardware wasn't *designed* for animation, that doesn't mean we can't put it to work in that capacity. One lesson that any good PC graphics or assembler programmer learns quickly is that it's what the PC's hardware *can* do—not what it was intended to do—that's important.

(By the way, if I were to pick one aspect of the PC to dump on, it would be sound, not animation. The PC's sound circuitry really is lousy, and it's hard to understand why that should be, given that a cheap sound chip—which even the PC*jr* had—would have changed everything. I guess IBM figured "serious" computer users would be put off by a computer that could make fun noises.)

Anyway, my point is that the PC's animation capabilities are pretty good. There's a trick, though: You can push the EGA and VGA (henceforth referred to as the EVGA) to their animation limits only by stretching your mind a bit and using some unorthodox approaches to animation. In fact, stretching your mind is the key to producing good code for *any* task on the PC—that's the topic of my book *The Zen of Assembler,* due out this summer from Scott, Foresman and Company. For most software, however, it's not fatal if your code isn't excellent: There's slow but functional software all over the place. When it comes to EVGA animation, though, you won't get to first base without a clever approach.

So, what clever approaches do I have in mind? All sorts. The resources of the EVGA are many and varied, and they can be applied and combined in hundreds of ways to produce effective animation. For example, refer back to the very first "On Graphics" (*PJ* Volume 5.1, January/February 1987) for a discussion of page flipping. Or look at the July 1986 issue of *PC Tech Journal,* which describes the basic block-move animation technique, or the August 1987 issue of *PC Tech Journal,* which shows a software-sprite scheme built around the EGA's vertical interrupt and the AND-OR image-drawing technique. Or look over the last nine installments of "On Graphics," which contain literally dozens of graphics tips and tricks that can be applied to animation.

This month we're going to add another entry to the list. We're going to take advantage of the bit-plane architecture and color palette of the EVGA to develop an animation architecture that can handle several overlapping images with terrific speed and with virtually perfect visual quality. This technique produces no overlap effects or flicker and allows us to use the fastest possible method to draw images: the **rep movs** instruction.

As with any technique on the PC, there are tradeoffs involved with bit-plane animation. While bit-plane animation is extremely attractive as far as performance and visual quality are concerned, it is somewhat limited. It supports only four colors plus the background color at any one time, each image must consist of only one of the four colors, and images of the same color preferably should not intersect.

It doesn't much matter if bit-plane animation isn't perfect for all applications, though. The real reason for showing you bit-plane animation is to bring home the reality that the EVGA is a

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

complex adapter with many resources and that you can do remarkable things if you understand those resources and come up with creative ways to put them to work at specific tasks.

Bit Planes: The Basics

The underlying principle of bit-plane animation is extremely simple. The EVGA has four separate bit planes in mode 10H. Plane 0 normally contains data for the blue component of pixel color, plane 1 normally contains green pixel data, plane 2 contains red pixel data, and plane 3 contains intensity pixel data. We're going to mix that up a bit in a moment, so we'll simply refer to them as planes 0, 1, 2, and 3.

Each bit plane can be written to independently. The contents of the four bit planes are used to generate pixels, with the four bits that control the color of each pixel coming from the four planes. However, the bits from the planes go through one stage on the way to becoming pixels: They're used to look up a six-bit color from one of the sixteen palette registers. Figure 1 shows how the bits from the four planes feed into the palette registers to select the color of each pixel.

The preceding paragraph is nothing more than a restatement of basic EVGA operation. I don't have space to recap that information in detail; refer back to the first ten installments of "On Graphics" if you're getting lost. For those of you who are still with me, take a good look at Figure 1. Any light bulbs going on over your head yet? Pity. Well, let's continue.

The general problem with EVGA animation is that it's complex and time-consuming to manipulate images that span the four planes (as most do) and that it's hard to avoid interference problems when images intersect, since those images share the same bits in display memory. Since the four-bit planes can be written to and read independently, it should be apparent that if we could come up with a way to display images from each plane independently of whatever images are stored in the other planes, we would have four sets of images that we could manipulate very easily. There would be no interference effects between images in different planes, since images in one plane wouldn't share bits with images in another plane. What's more, since all the bits for a given image would reside in a single plane, we could do away with the cumbersome programming of the EVGA's complex hardware that is needed to manipulate images that span multiple planes.

All in all, it would be a good deal if we could store each image in a single plane, as shown in Figure 2. However, a problem arises when images in different planes overlap, as shown in Figure 3. The combined bits from overlapping images generate new colors, so the overlapping parts of the images don't look like they belong to either of the two images. What we really want, of course, is for one of the images to appear to be in front of the other. It would be better yet if the rearward image showed through transparent (background-colored) parts of the forward image. Can we do that? You bet.

Stacking the Palette Registers

Suppose that instead of viewing the four bits per pixel coming out of display memory as selecting one of sixteen colors, we view those bits as selecting one of *four* colors. If the bit from plane 0 is 1, that would select color 0 (say, red). The bit from plane 1 would select color 1 (say, green), the bit from plane 2 would select color 2 (say, blue), and the bit from plane 3 would select color 3 (say, white). Whenever more than one bit is 1, the 1 bit from the lowest-numbered plane

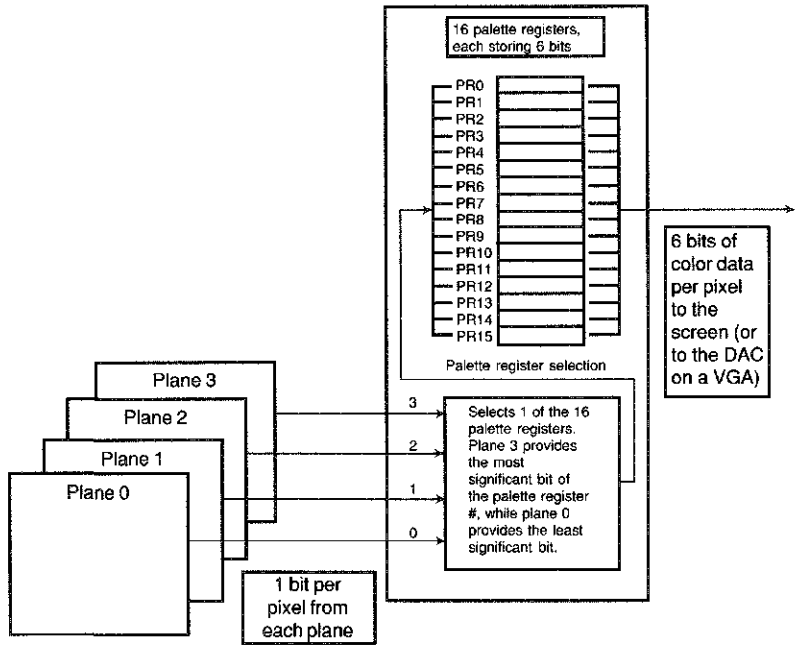


Figure 1. A summary of the path by which four bits of video data (one bit from each of the four planes) become a six- bit pixel color

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Bit Value For Plane 3 2 1 0	Palette Register	Register Setting
0 0 0 0	0	00h (black)
0 0 0 1	1	3Ch (red)
0 0 1 0	2	3Ah (green)
0 0 1 1	3	3Ch (red)
0 1 0 0	4	39h (blue)
0 1 0 1	5	3Ch (red)
0 1 1 0	6	3Ah (green)
0 1 1 1	7	3Ch (red)
1 0 0 0	8	3Fh (white)
1 0 0 1	9	3Ch (red)
1 0 1 0	10	3Ah (green)
1 0 1 1	11	3Ch (red)
1 1 0 0	12	39h (blue)
1 1 0 1	13	3Ch (red)
1 1 1 0	14	3Ah (green)
1 1 1 1	15	3Ch (red)

Table 1

would determine the color, and 1 bits from all other planes would be ignored. Finally, the absence of any 1 bits at all would select the background color (say, black).

That would give us four colors and the background color. It would also give us nifty image precedence, with images in plane 0 appearing to be in front of images from the other planes, images in plane 1 appearing to be in front of images from planes 2 and 3, and so on. It would even give us transparency, where rearward images would show through holes within and around the edges of images in forward planes. Finally, and most important, it would meet all the criteria needed to allow us to store each image in a single plane, letting us manipulate the images very quickly and with no reprogramming of the EVGA's hardware other than the few **out** instructions required to select the plane to which we want to write.

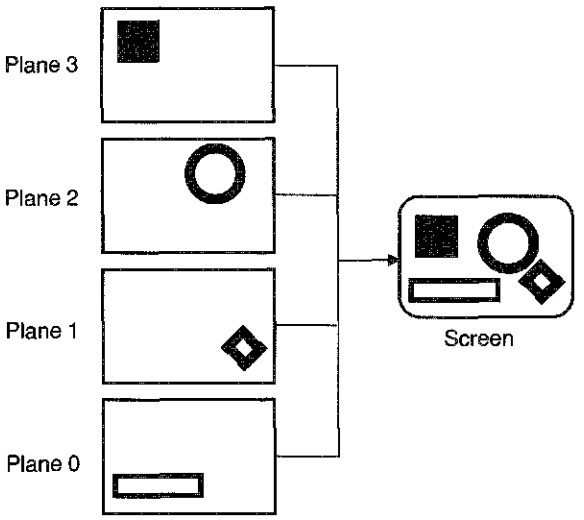


Figure 2. It would be useful if we could store each image in a single plane. The images would appear together on the screen but could be manipulated without the cumbersome programming of the EVGA's complex hardware that is required for images that span multiple planes.

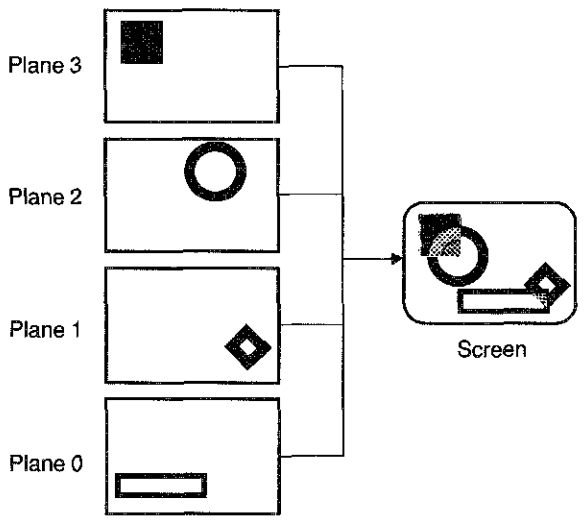


Figure 3. If we store each image in a single plane, undesirable color effects result when the images overlap.

Which leaves only one question: How do we get this magical pixel-precedence scheme to work? As it turns out, it's a simple matter of reprogramming the palette registers. All we need to do is reprogram the palette registers so that the 1 bit from the plane with the highest precedence determines the color. Table 1 lists the palette RAM settings for the colors described above.

Remember that the four-bit values coming from display memory select which palette register provides the actual pixel color. Given that, it's easy to see that the rightmost 1 bit of the four bits coming from display memory in the table selects the pixel color. If the bit from plane 0 is 1, then the color is red, no matter what the other bits are, as shown in Figure 4. When the bit from plane 0 is 0 and the bit from plane 1 is 1, the color is green, and so on for planes 2 and 3. In other words, with the palette register settings given in the table, we instantly have exactly what we want, which is an approach that keeps images in one plane from interfering with images in other planes while providing precedence and transparency.

Seems almost too easy, doesn't it? It works beautifully, however, as we'll see very shortly. First, though, I'd like to point out that there's nothing sacred about plane 0 having precedence. We could rearrange the palette register

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

INGRAF 2.00

Full Source code $295.00

The multi-device **GRAPHICS** library for scientific/ Engineering and Busines Graphics applications.**INGRAF** contains over **100 routines**.

Features include:

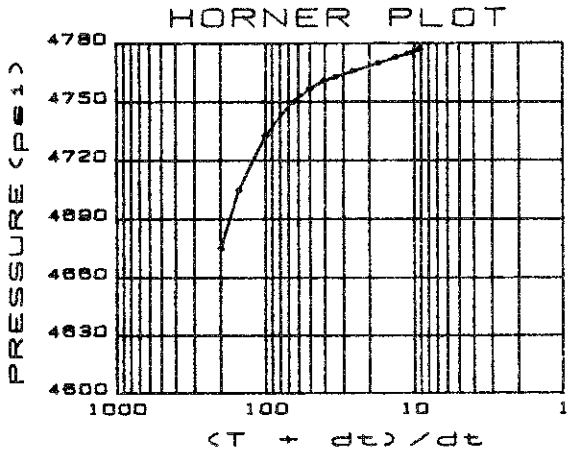
- *Linear, Semilog, Log/Log, Polar plots.*
- *Axes and Grids with tick marks and labels.*
- *Bar, Pie, Smith charts.*
- *Markers, Line types, Curves, Arcs, Circles, Ellipses.*
- *Filling and Hatching of bounded areas.*

Supports:

- *CGA, EGA, VGA, HERCULES Graphics cards.*
- *HPGL, DM/PL pen plotters.*
- *HP LaserJet, HP DeskJet, Epson, Okidata printers.*

Languages Supported:

- **C**
- **FORTRAN-77**
- **BASIC**
- **PASCAL**



Compilers Supported:

C	FORTRAN-77	BASIC	PASCAL
Microsoft C	MS FORTRAN	Quick Basic	MS Pascal
Turbo C	RM FORTRAN	Turbo Basic	Turbo Pascal
Lattice C	Lahey F77L		
MWC-86	IBM Prof. FORTRAN		

FORTLIB 4.00

Full Source code $150.00

The **FORTRAN-77** Enhancement library for **IBM-PC (MS-DOS/PC-DOS)** and compatibles. The library contains well over **200 routines**. Most routines are written in assembly for speed and compact code.

Features include:

- *Keyboard handling.*
- *Video setup and control.*
- *Date and time functions.*
- *Disk directory and file handling.*
- *General I/O port functions.*
- *Memory PEEK and POKE functions.*
- *Data sorting.*
- *Extensive string handling.*
- *Data conversion routines.*
- *Character classification functions.*
- *RS-232 service routines.*
- *Mepory manipulation,*
- *printer and system service routines.*
- *and many miscellaneous routines.*

GRAFLIB 4.00 Full Source code $175.00

Video Graphics Library for **IBM-PC** and compatibles. The library supports **CGA, EGA, VGA** and **HERCULES** Graphics cards. There are over 50 routines for drawing **Arcs, Circles, Line drawing, Axes, Labeling of text, Curves, Line types, Markers** to name a few.

The library is available in C & **FORTRAN-77** bindings.

PLOTHP 4.00 Full Source code $175.00

Plotting routines for **HPGL(Hewlett-Packard)** compatible plotters. The Library contains over **60 routines** for drawing **Arcs, Circle, Curves, Linear, Semilog, Log/Log** plots. **Axes drawing with tick marks, Text labeling, Scaling of plots** is simple with **PLOTHP** routines. The library includes routines for specifying **Line types, Marker types, Text direction** and **Orientaton.**

The Library is available in C & **FORTRAN-77** bindings.

For more information and a detailed brochure please call or write to:

Sutrasoft (713) 491-2088

10506 Permian Dr.
P.O. Box 1733
Sugar Land, TX 77487-1733

Number **253** on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

settings so that any plane had the highest precedence, followed by the other planes in any order. I've chosen to give plane 0 the highest precedence only because it seems simplest to think of plane 0 as appearing in front of plane 1, which is in front of plane 2, which is in front of plane 3.

Bit-Plane Animation in Action

The listing that begins on page 28 shows bit-plane animation in action. The listing animates thirteen rather large images (each thirty-two pixels on a side) over a complex background at a good clip—*even* on a PC. Five of the images move very quickly, while the other eight bounce back and forth at a steady pace.

I'm not going to discuss the listing in detail. For one thing, that would take too much space. For another, the code is very thoroughly commented and should speak for itself. In any case, most of the individual components of the listing—the Map Mask register, mode sets, word versus byte **out** instructions to the EVGA—have come up in earlier installments of "On Graphics." Do notice, however, that the listing sets the palette exactly as I described earlier. This is accomplished by passing a pointer to a seventeen-byte array (one byte for each of the palette registers and one byte for the border color) to the BIOS video interrupt (INT 10H), function 10H, subfunction 1.

There's a whole lot of animation going on in the listing. What's more, the animation is virtually flicker free, partly thanks to bit-plane animation and partly because images are never really erased but are simply overwritten. The principle behind the animation is that of redrawing each image with a blank fringe around it when it moves so that the blank fringe erases the part of the old image that the new image doesn't overwrite.

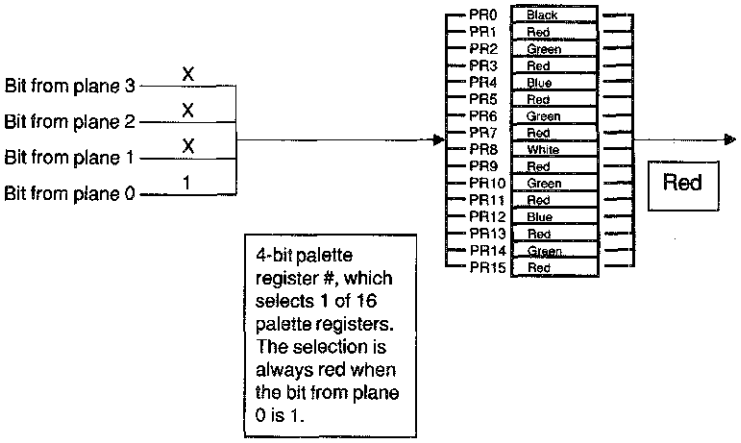


Figure 4. When the palette registers are set up for bit-plane animation, with plane 0 given the highest precedence and mapped to red, then whenever a pixel's plane 0 bit is 1, red is displayed, no matter what the values of the bits from the other planes.

GRAPHICS

Break Out of the Text Barrier with your Herc, CGA, EGA, or VGA

PCX PROGRAMMER'S TOOLKIT $125
Over 35 routines to display, save, print PCX bitmapped graphics - Fast, easy to use - Display windows or full screen - Image library manager - use images from Paintbrush, Scanners, Clip-Art, or Capture Screens - No Royalties !

PCX TEXT $99
Display text in any graphics mode - Includes 12 fonts - Create custom fonts with dual font editor - User input routines - Font scaling - Additional fonts available.

PCX F/X $99
Add Special Effects ('FX')! Fade, wipe, push, roll, slide, split, crush, spiral, explode your graphics! Palette manipulation - Animation - Sound Effects

All packages support 12 compilers for BAS, PAS, C, FOR, ASM and Clipper

G·E·N·U·S
MICROPROGRAMMING
11315 Meadow Lake • Houston, Texas 77077 • (713) 870-0737
800•227•0918

Number 261 on the Reader Service Card

"The cost involved, in writing just one TurboGeometry routine, is more than the price of the library itself".

Are you programming or planning to program in CAD/CAM or graphics applications? Many hours, even days can be spent in writing and debugging geometric routines. ***TurboGeometry*** library can relieve you of these time consuming tasks. There are over 150 routines in the library. Some of the areas covered are: Equations of lines, circles and planes • Intersection of lines, circles, arcs and planes • 2 and 3 dimensional transforms • Decomposition of polygons • Hidden line removal • Volume and area calculations • Perspectives • Polygon line clipping • and many more! Supported by example programs and a 400 page manual. Source code included. 30 day guarantee. Need IBM PC or Comp/Macintosh. MS/PC DOS 2.0+. Available in Turbo Pascal, Turbo C, Microsoft C and Turbo Pascal Macintosh. $149.95 plus $5.00 S&H. Foreign orders add $16.00 for S&H Air. VISA, MasterCard, M/O, Chk. To order or request information call **214-423-7288 / Fax# 214-423-4465** or write to:

TurboGeometry Library

Disk Software, Inc. • 2116 E. Arapaho Rd., Suite 487 Richardson, TX USA 75081
(To order toll free, call **1-800-635-7760** in US only).

Number 229 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

(For details on this sort of animation, see the aforementioned *PJ* Volume 5.1 and *PC Tech Journal* July 1986 articles.)

Better yet, the red images take precedence over the green images, which take precedence over the blue images, which take precedence over the white backdrop, and all obscured images show through holes in and around the edges of images in front of them. In short, The listing accomplishes everything we wished for earlier in an animation technique.

If you can, run the listing. The animation may be a revelation to those of you who are used to weak, slow animation on PCs or EVGAs. Bit-plane animation makes the PC look an awful lot like—Dare I say it?—a game machine. If you're running the listing on an AT or a faster computer (and considering the sophistication of *PJ*'s readership, I'd say that's a pretty safe bet), you'll have to crank up the **DELAY** equate at the start of the listing to slow things down. Even on a stock AT, the listing runs much too fast without a substantial delay (although it does look rather interesting at warp speed). We should all have such problems, eh? In fact, we could easily increase the number of animated images past twenty on an AT.

Bit-plane animation does have serious limitations, which we'll get to in a second. One limitation that is *not* inherent to bit-plane animation but is simply a shortcoming of the listing is somewhat choppy horizontal motion. In the interests of both clarity and keeping the listing to a reasonable length, I decided to byte-align all images horizontally. This saved the many tables needed to define the seven non–byte-aligned rotations of the images as well as the code needed to support rotation. Unfortunately, it also meant that the smallest possible horizontal movement was eight pixels (one byte of display memory), which is far enough to be noticeable at certain speeds. The situation is, however, easily correctable with the additional rotations and code, as described in the articles mentioned above. Vertically, where there is no byte-alignment issue, the images move four or six pixels at a times, resulting in considerably smoother animation.

The addition of code to support rotated images also opens the door to support for internal animation, where the appearance of a given image changes over time to suggest that the image is an active entity. For example, propellers could whirl, jaws could snap, and jets

POWER GRAPHICS FOR YOUR PC!

MetaWINDOW

NO ROYALTIES!

PC TECH JOURNAL
Product of the Month
"Awarded for excellence in product design and development"

Developing a graphics application? Move to the peak of performance and graphics functionality with MetaWINDOW! Built on state of the art "object oriented" display flexibility, now you can take advantage of powerful advanced graphics capabilities like those used on the Macintosh, Sun and Apollo workstations. Using MetaWINDOW you can break the bottleneck between limited function low-level graphic primitive libraries and slow, bulky hard-coded window managers.

MetaWINDOW provides an advanced set of graphic drawing functions, dynamic runtime support for a full range of graphic peripherals, plus unmatched performance for designing multi-window desktop applications.

Awarded Top Honors!

PC Tech Journal, Jul '85
Product of the Month - *"... a technological tour de force for fast PC graphics." "... one high-powered piece of artillery."*

Computer Language, Oct '87 - *"... surprisingly powerful for the cost." Independently benchmarked the fastest of all graphics systems reviewed - faster than Halo, over 3 times faster than GSS*CGI.*

Turbo Technix, Feb '88 *"... Customer support is excellent. I've called Metagraphics several times and have gotten prompt, accurate help each time. Metagraphics has a winner with TurboWINDOW."*

Superior Functionality!

- Macintosh QuickDraw function superset - *simplifies porting PC & Mac applications.*
- PostScript & X-Window compatible drawing attributes - the defacto graphics *standard for PC X-Window servers.*
- Supports both bitmap and outline fonts - *dynamically face fonts for bold, italic, underline or strike-out effects.*
- Full 16 function "rasterOp" flexibility - *easily write, erase, rubberband or drag: lines, text, icons, bit images and complex objects.*
- Dynamic recognition and run-time support for 80+ popular graphics cards - *eliminates multitudes of clumsy device drivers and bothersome configuration sessions - true device independence at last!*
- Virtual images - *create and manipulate graphic images in local or expanded memory up to 8 megabytes in size.*
- *... and the list goes on!*

Rated #1 by Users!

J. Kayser, CMSI -
"MetaWINDOW is the best graphics package I've seen. Since I bought my PC, I've spent over $2500 on software. I've bought bad software, mediocre software, and good software. There's not much software I call excellent, but MetaWINDOW is excellent!"

David Gray, LUCAS LABS -
"I would also like to take this time to compliment you on providing such a fantastic graphics package. We have used several packages previously including Turbo graphics and Halo graphics and found none that compare to the versatile, high performance graphics package that your company offers. We have also found your technical support staff to be one of the best we have ever dealt with."

John Weaver, Chairperson
Department of Mathematical Sciences
WEST CHESTER UNIVERSITY -
"We have been using MetaWINDOW for our graphics instruction for the past several semesters and are pleased with the package. This past summer we upgraded one system to MetaWINDOW version 3 and were so impressed that we want to upgrade all systems prior to the Spring semester. This is a first class package and I think it is the best for teaching computer graphics."

Reach For The Future!

MetaWINDOW - $195
Advanced Graphics Development Toolkit
MetaWINDOW comes complete with language bindings for 20 popular C, Pascal and Fortran compilers, plus dynamic runtime support for over 80 graphics adaptors and input devices.
4 disks, 3 260 page manuals

TurboWINDOW/C - $95
TurboWINDOW/Pascal - $95
QuickWINDOW/C - $95
All the features of MetaWINDOW for Borland Turbo C, Turbo Pascal or Microsoft C.

NEW - FontWINDOW - $95
MetaWINDOW font/icon editor plus 100 fonts.

NEW - MetaWINDOW/Premium - $495
Power graphics for new high-res, high performance TI34010 based graphics cards!
(Licensing required for redistribution of MetaWINDOW/Premium)

30 Day Money Back Guarantee
ORDER TODAY! 800-332-1550
For information or in CA call 408-438-1550

METAGRAPHICS
SOFTWARE CORPORATION
269 Mount Hermon Road
Scotts Valley, CA 95066

Number 143 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

could flare. Bit-plane animation with byte-aligned images and internal animation can look truly spectacular. Perhaps someday I can wangle space for an extra-long "On Graphics" and put all the animation pieces together in a single listing. It's a sight worth seeing, particularly for those who doubt the PC's worth when it comes to animation.

Limitations of Bit-Plane Animation

As I've said, bit-plane animation is not perfect. For starters, bit-plane animation can be used only in the EVGA's planar modes: 0DH, 0EH, 0FH, 10H, and 12H. Also, the reprogramming of the palette registers that provides image precedence reduces the available color set from the normal sixteen colors to just five (one color per plane plus the background color). Worse still, each image can be drawn in only one of the four colors. Mixing colors within an image is not allowed, since the bits for each image are limited to a single plane and therefore can select only one color. Finally, all images of the same precedence must be the same color.

It is possible to work around the color limitations to some extent by using only one or two planes for bit-plane animation while reserving the other planes for multicolor drawing. For example, you could use plane 3 for bit-plane animation while using planes 0, 1, and 2 for normal eight-color drawing. The images in plane 3 would then appear to be in front of the eight-color images. If we wanted the plane-3 images to be yellow, we could set up the palette registers as shown in Table 2.

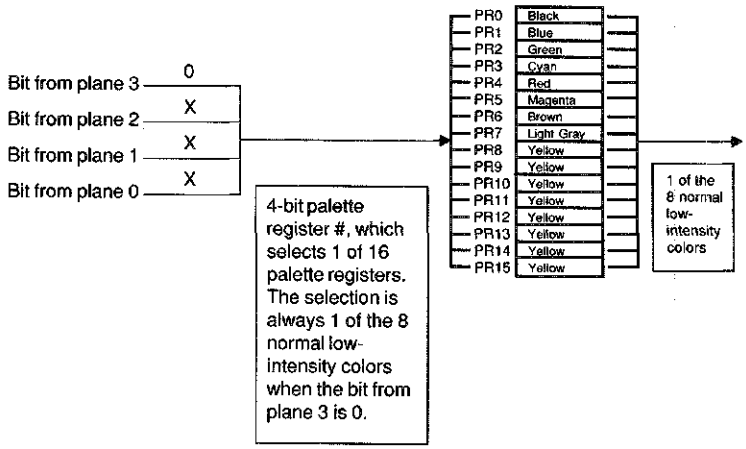


Figure 5. When the palette registers are set up as shown for bit-plane animation with one bit plane (plane 3) appearing in front of an eight-color backdrop, then whenever a pixel's plane-3 bit is 0, one of the eight normal low-intensity colors is selected by the three bits from the other planes.

As you can see, the color yellow is displayed whenever a pixel's bit from plane 3 is 1. This gives the images from plane 3 precedence while leaving us with the eight normal low-intensity colors for images drawn across the other three planes, as shown in Figure 5. Of course, this approach provides only one rather than three high-precedence planes, but that might be a good tradeoff for being able to draw multicolored images as a backdrop to the high-precedence images. For the right application, high-speed flicker-free plane-3 images moving in front of an eight-color backdrop could be a potent combination indeed.

Another limitation of bit-plane animation is that it's best if images stored in the same plane never cross each other.

Palette Register	Register Setting
0	00h (black)
1	01h (blue)
2	02h (green)
3	03h (cyan)
4	04h (red)
5	05h (magenta)
6	14h (brown)
7	07h (light gray)
8	3Eh (yellow)
9	3Eh (yellow)
10	3Eh (yellow)
11	3Eh (yellow)
12	3Eh (yellow)
13	3Eh (yellow)
14	3Eh (yellow)
15	3Eh (yellow)

Table 2

People are talking about us.

When professional FORTRAN programmers develop or port large programs they use Lahey's F77L-EM/32 and F77L-EM/16, *PC Magazine's 1988 Technical Excellence Award Winners.* F77L-EM/32 is a fast 32-bit protected-mode compiler that accesses up to 4 gigabytes of memory on 80386s. F77L-EM/16 gives 80286 users the power to create 15 megabyte programs. These protected-mode FORTRANS include the features that have made them, and our F77L and Lahey Personal FORTRAN, market leaders: full ANSI 77 Standard, VAX and IBM VS extensions, fast compilation, comprehensive diagnostics, and a powerful debugger.

TECHNICAL EXCELLENCE PC MAGAZINE WINNER COMPILERS/LANGUAGES F77L-EM/16 AND F77L-EM/32 LAHEY COMPUTER SYSTEMS, INC. 1988

When people talk about FORTRAN the name mentioned most often is Lahey Computer Systems Inc

Contact us to discuss our products and your needs. (800) 548-4778
Lahey Computer Systems, Inc. P.O. Box 6091, Incline Village, NV 89450
Tel: (702) 831-2500 FAX: (702) 831-8123 Tlx: 9102401256

Number 44 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

When images do cross, the blank fringe around each image can temporarily erase the overlapped parts of the other image or images, resulting in momentary flicker. While that's not fatal, it certainly detracts from the rock-solid animation effect of bit-plane animation.

Not allowing images in the same plane to overlap is actually less of a limitation than it seems. Run the listing again. Unless you were looking for it, you'd never notice that images of the same color almost never overlap—there's plenty of action to distract the eye, and the trajectories of images of the same color are arranged so that they have a full range of motion without running into each other. The only exception is the chain of green images, which occasionally doubles back on itself when it bounces directly into a corner and reverses direction. Here, however, the images are moving so quickly that the brief moment during which one image's fringe blanks a portion of another image is noticeable only upon close inspection—and it's not particularly unaesthetic even then.

When a technique has such tremendous visual and performance advantages as bit-plane animation does, it behooves you to design your animation software so that the limitations of the animation technique don't get in the way. For example, you might design a shooting gallery game with all the images in a given plane marching along in step in a continuous band. Since the images could never overlap, you could use bit-plane animation.

Shearing and Page Flipping

As the listing runs, you may occasionally see an image shear, with the top and bottom parts of the image briefly offset. This is a consequence of drawing an image directly into memory as that memory is being scanned for video data. Occasionally, the CRT controller scans a given area of display memory for pixel data just as the program is changing that same memory. If the CRT controller scans memory faster than the CPU can modify that memory, then the CRT controller can scan out the bytes of display memory that have been already been changed, pass the point in the image that the CPU is currently drawing, and start scanning out bytes that haven't yet been changed. The result is mismatched upper and lower portions of the image.

If the CRT controller scans more slowly than the CPU can modify memory (likely with a 386, a fast VGA, and narrow images), then the CPU can pass the CRT controller, with the same net result of mismatched top and bottom parts of the image as the CRT controller scans out first unchanged bytes and then changed bytes. Basically, shear will occasionally occur unless the CPU and CRT proceed at exactly the same rate, which is most unlikely. Shear is more noticeable when there are fewer but larger images, since it's more apparent when a larger screen area is sheared and because it's easier to spot one out of three large images momentarily shearing than one out of twenty small images.

Image shear isn't terrible—I've written and sold several games in which images occasionally shear, and I've never heard anyone complain—but neither is it ideal. One solution is page flipping, in which drawing is done to a nondisplayed page of display memory while another page of display memory is shown on the screen. When the drawing is finished, the newly drawn part of display memory is made the displayed page so that the new screen is made visible all at once, with no shearing or flicker. The other

Quality Programming Tools from Ithaca Street Software, Inc.

Font-Tools

Provides a total solution to your graphics screen font needs. Create and edit fonts with our full-featured editor. Additional utilities allow you to modify, view, download, or format convert fonts. We provide over 60 high quality sample fonts in a variety of styles and sizes. Display fonts from a C graphics application with the included function library, which offers over 40 functions for text drawing.

Single User $99.95
Developer's Ed. $149.95

Baby Driver

When it comes to graphics printer support, this is the only product of its kind. More than 150 dot matrix, inkjet, and laser printers supported. Print from screen areas or memory buffers with full control of scaling, orientation, color rendition, and more. Function library is linked with your application, adds only 10K. Includes PCL functions for desktop publishing applications. PDT editor allows you to add your own support.

Single User $99.95
Developer's Ed. $199.95

Icon-Tools/Plus

Icons and fonts may seem similar, but their uses are typically quite different. That's why you need a product like Icon-Tools. We provide a versatile icon editor, utility programs, sample icon files with more than 500 icons, and a function library to draw and use icons from a graphics application. The functions provide menu and mouse support and sound and animation effects to cover a wide range of programming needs. A nifty product.

Single User $99.95
Developer's Ed. $125.00

*1145 Ithaca Drive, Boulder, CO 80303 * Phone (303) 494-8865 * Fax (303) 494-7802*

Number 226 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

page is then drawn to, and when the drawing is complete the display is switched back to that page.

Page flipping can be used in conjunction with bit-plane animation, although page flipping does diminish some of its unique advantages. Page flipping produces animation of the highest visual quality, whether bit-plane animation is used or not. There are a few drawbacks to page flipping, however:

- Page flipping requires two display memory buffers, one to draw in and one to display at any given time. Unfortunately, in mode 12H there just isn't enough memory for two buffers, so page flipping is not an option in that mode.
- Page flipping requires that you keep the contents of both buffers up to date, which can necessitate a good deal of extra drawing.
- Page flipping requires that you wait until you're sure the page has flipped before you start drawing to the other page. Otherwise, you could end up modifying a page while it's still being displayed, defeating the whole purpose of page flipping. Waiting for pages to flip takes time and can slow overall performance significantly. What's more, it's sometimes difficult to be sure when the page has flipped, since not all EVGA clones implement the EVGA status bits and page-flip timing identically.

To sum up, bit-plane animation by itself is very fast and looks good. In conjunction with page flipping, bit-plane animation looks a little better but is slower, and the overall animation scheme is more difficult to implement and perhaps a bit less reliable on some computers.

In Conclusion

Bit-plane animation is neat stuff. Heck, good animation of *any* sort is fun, and the PC is as good a place as any (well, almost any) to make people's jaws drop. Don't let anyone tell you that you can't do good animation on the PC. You can—*if* you stretch your mind to find ways to bring the full power of the EGA and VGA to bear on your applications. Bit-plane animation isn't for every task; neither are page flipping, exclusive ORing, pixel panning, or any of the host of other techniques we've discussed. One or more tricks from that grab bag should give you what you need, though, and the bigger your grab bag, the better your programs. What we're doing in "On Graphics" is making the biggest grab bag of graphics tricks we can. Let me know if you've got some nifty tricks to add or if there's anything you'd like to see covered.

See you next time.

Michael Abrash is a Senior Software Engineer for Orion Instruments, Inc., of Redwood City, California, where he is working on the design of advanced PC-based test instrumentation. Readers may write to Michael at P.O. Box 39051, Mountain View, CA 94039, or drop him a line via MCI Mail, user name MABRASH.

Code follows. ♦

EVERTRAK

SECURITY

If you HATE Copy Protection, but still want to protect your program from hackers and reverse engineering, EVERTRAK allows you to:

* Shut down DEBUG systems and disassemblers.
* Restrict use by date-range and media-type.
* Embed a secure serial number string.
* Track unauthorized copies back to their source.

EVERTRAK is totally invisible to your end user and uses the latest encryption technology.

EVERTRAK supports IBM & Compatibles using DOS 2.0 or later, sells for $295.00 and has a 30 day money back guarantee.

AZ

Az-Tech Software, Inc.
305 East Franklin
Richmond, MO 64085

(800) 227-0644 **(816) 776-2700**

Number **51** on the Reader Service Card

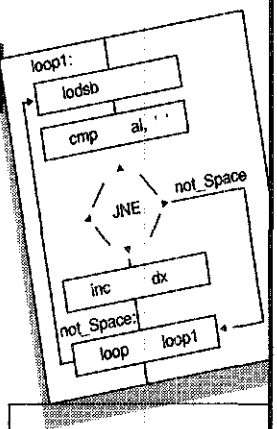
ASMFLOW $99.95

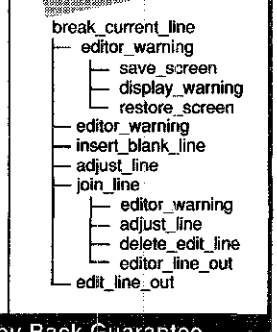
AT LAST!

YOU CAN STEP BACK AND TAKE A LOOK AT YOUR CODE

FLOW CHART AND ANALYZE YOUR ASSEMBLY LANGUAGE SOURCE CODE

- Flow Charts
- Tree Diagrams
- Stack Sizing
- Register Analysis
- CPU Timing Analysis
- Procedural X-Reference
- 8088/87 to 80386/387
- Context-Sensitive Help
- Menu/Batch/Command line Operation
- MASM 5.1 Compatible





QUANTUM SOFTWARE

19855 Stevens Creek Blvd, Suite 154
Cupertino, CA 95014
(408) 244-6826 – Free Demo

VISA • MC 30 - Day Money Back Guarantee

Number **252** on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

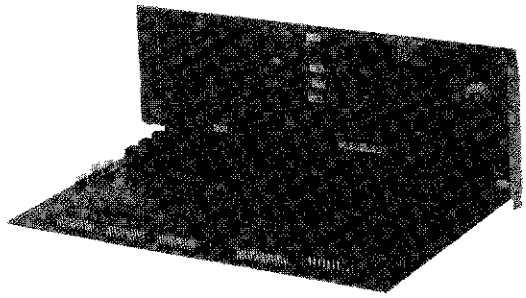
VERY HIGH PERFORMANCE

Processors, Memory, and Display Adapters

The X24 High performance processor

- 12 or 16 MHz 80286 with NO WAIT STATES!
- Small size ("XT" height and length) passive bus design
- 1 to 4 Mbyte 0 wait state dynamic memory
- Fully "AT" compatible Award BIOS
- Runs DOS versions 2.2 and later, Xenix and OS/2

The X24 combines the best of motherboard and backplane designs in a 100% AT compatible system. Incorporating a 16 MHz 80286, the X24 processor is designed to operate with the PC Tech Advanced System Motherboard, which contains the peripheral interfaces (hard disk, floppy disk, two serial ports and a parallel port). The X24 processor can also be used with other totally passive bus backplanes. Most critical components including the microprocessor and up to 4 megabytes of fast memory are contained on a single PC size plug-in card. This allows the processor and main system memory to be serviced or upgraded without disturbing other peripherals such as serial ports and disk drives.



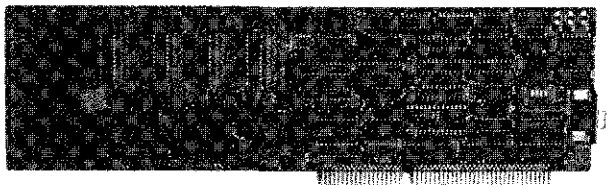
PC Tech X24 and ASMB

The PC Tech Advanced System Motherboard

- Built in "IDE" interface for AT interface type hard drives
- Fully AT compatible floppy disk support for 3.5", 5.25" drives, capacities of 360k, 1.2m and 1.44m
- Two serial ports and one parallel port
- 8 total expansion slots PC/XT/AT compatible (4 slots have 32 bit bus)

The PC Tech Advanced System Motherboard is designed to complement PC Tech's X24 and X32 high performance processor cards. It contains the mass storage interfaces necessary for a complete system, plus the basic I/O required in most systems. Extra care has been given to FCC compliance by design.

34010 Monochrome Graphics Adapter II



PC Tech Mono-II

- Up to 384k bytes display memory
- Up to 2 Megabytes program memory
- Software is RAM based, allowing complete operating software replacement and timing re-programming from the host bus
- 34010 program loader included. Assembler, debugger, and C compiler available.
- Full hardware and software CGA, MDA and Hercules emulation
- Single bit shared memory bit-map with optional resolution up to 2048 x 1536 (736 x 1008 standard)
- Very high resolution COLOR version available
- Custom 34010 software development available

The TMS34010 is a true general purpose graphics processor. PC Tech makes the total processing power of the 34010 available to both programmers and end users. Our 34010 Monochrome Graphics Adapter is designed to allow programming from the PC/XT/AT host bus. You can completely replace our 34010 software with yours to directly harness the incredible image processing power of the TMS 34010 for your application. We make a complete set of development tools available, including an assembler, C compiler, program loader, 34010 debugger, and PC interface tracer/debugger. Our standard product includes support for extended CGA, MDA and Hercules emulation as well as a host addressable graphics bit-map. We also support and recommend the DGIS graphics interface standard (from Graphic Software Systems) for applications development as an alternative to native 34010 software development. Ready to run drivers are available for most major applications software packages as well.

Custom Designs Available

PC Tech will license most products for non-exclusive manufacture. We will also customize any of our designs to better meet your needs on our in-house CAD systems. All of our standard products are available in private label versions.

About PC Tech

PC Tech has been designing, manufacturing and marketing high performance PC related products for over three years. Our standard product line includes processor, memory, and video products. All products are designed, manufactured and supported in our Lake City, Minnesota facilities.

Designed, Sold and Serviced By:

PTech

907 N. 6th St., Lake City, MN 55041
(612) 345-4555 • (612) 345-5514 (FAX)

High resolution fractal produced on the PC Tech COLOR 34010

Number **208** on the Reader Service Card

PC, XT, AT, DGIS, Hercules, and Award BIOS are trademarks or registered trademarks of their respective companies.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

All code in this issue is available on 360K, MS-DOS disk.
ORDER BY PHONE — VISA/MC orders call toll free 800/234-0386.
Mail order information on pages 8 and 88.

Code from "EGA and VGA Animation" by Michael Abrash

1

```
; Program to demonstrate bit-plane animation. Performs
; flicker-free animation with image transparency and
; image precedence across four distinct planes, with
; thirteen 32x32 images kept in motion at once.
;
;
; Set to higher values to slow down on faster computers.
; 0 is fine for a PC. 500 is a reasonable setting for an AT.
; Slowing animation further allows a good look at
; transparency and the lack of flicker and color effects
; when images cross.
;
SLOWDOWN        equ     0
;
; Plane selects for the four colors we're using.
;
RED     equ     01h
GREEN   equ     02h
BLUE    equ     04h
WHITE   equ     08h
;
VGA_SEGMENT     equ     0a000h  ;mode 10h display memory
                                ; segment
SC_INDEX        equ     3c4h    ;Sequence Controller Index
                                ; register
MAP_MASK        equ     2       ;Map Mask register index in
                                ; Sequence Controller
SCREEN_WIDTH    equ     80      ;# of bytes across screen
SCREEN_HEIGHT   equ     350     ;# of scan lines on screen
WORD_OUTS_OK    equ     1       ;set to 0 to assemble for
                                ; computers that can't
                                ; handle word outs to
                                ; indexed VGA regs
;
stack   segment para stack 'STACK'
        db      512 dup (?)
stack   ends
;
; Complete info about one object that we're animating.
;
ObjectStructure struc
Delay           dw      ?       ;used to delay for n passes
                                ; through the loop to
                                ; control animation speed
BaseDelay       dw      ?       ;reset value for Delay
Image           dw      ?       ;pointer to drawing info
                                ; for object
XCoord          dw      ?       ;object X location in pixels
XInc            dw      ?       ;# of pixels to increment
                                ; location by in the X
                                ; direction on each move
XLeftLimit      dw      ?       ;left limit of X motion
XRightLimit     dw      ?       ;right limit of X motion
YCoord          dw      ?       ;object Y location in pixels
YInc            dw      ?       ;# of pixels to increment
                                ; location by in the Y
                                ; direction on each move
YTopLimit       dw      ?       ;top limit of Y motion
YBottomLimit    dw      ?       ;bottom limit of Y motion
PlaneSelect     db      ?       ;mask to select plane to
                                ; which object is drawn
                db      ?       ;to make an even # of words
                                ; long, for better 286
                                ; performance (keeps the
                                ; following structure
                                ; word-aligned)
ObjectStructure ends
```

2

```
;
Data    segment word 'DATA'
;
; Palette settings to give plane 0 precedence, followed by
; planes 1, 2, and 3. Plane 3 has the lowest precedence (is
; obscured by any other plane), while plane 0 has the
; highest precedence (displays in front of any other plane).
;
Colors  db      000h ;background color=black
        db      03ch ;plane 0 only=red
        db      03ah ;plane 1 only=green
        db      03ch ;planes 0&1=red (plane 0 priority)
        db      039h ;plane 2 only=blue
        db      03ch ;planes 0&2=red (plane 0 priority)
        db      03ah ;planes 1&2=green (plane 1 priority)
        db      03ch ;planes 0&1&2=red (plane 0 priority)
        db      03fh ;plane 3 only=white
        db      03ch ;planes 0&3=red (plane 0 priority)
        db      03ah ;planes 1&3=green (plane 1 priority)
        db      03ch ;planes 0&1&3=red (plane 0 priority)
        db      039h ;planes 2&3=blue (plane 2 priority)
        db      03ch ;planes 0&2&3=red (plane 0 priority)
        db      03ah ;planes 1&2&3=green (plane 1 priority)
        db      03ch ;planes 0&1&2&3=red (plane 0 priority)
        db      000h ;border color=black
;
; Image of a hollow square.
; There's an 8-pixel-wide blank border around all edges
; so that the image erases the old version of itself as
; it's moved and redrawn.
;
Square  label   byte
        dw      48,6    ;height in pixels, width in bytes
        rept    8
        db      0,0,0,0,0,0     ;top blank border
        endm
        .radix  2
        db      0,11111111,11111111,11111111,11111111,0
        db      0,11111111,11111111,11111111,11111111,0
        db      0,11111111,11111111,11111111,11111111,0
        db      0,11111111,11111111,11111111,11111111,0
        db      0,11111111,11111111,11111111,11111111,0
        db      0,11111111,11111111,11111111,11111111,0
        db      0,11111111,11111111,11111111,11111111,0
        db      0,11111111,11111111,11111111,11111111,0
        db      0,11111111,00000000,00000000,11111111,0
        db      0,11111111,00000000,00000000,11111111,0
        db      0,11111111,00000000,00000000,11111111,0
        db      0,11111111,00000000,00000000,11111111,0
        db      0,11111111,00000000,00000000,11111111,0
        db      0,11111111,00000000,00000000,11111111,0
        db      0,11111111,00000000,00000000,11111111,0
        db      0,11111111,00000000,00000000,11111111,0
        db      0,11111111,00000000,00000000,11111111,0
        db      0,11111111,00000000,00000000,11111111,0
        db      0,11111111,00000000,00000000,11111111,0
        db      0,11111111,00000000,00000000,11111111,0
        db      0,11111111,00000000,00000000,11111111,0
        db      0,11111111,00000000,00000000,11111111,0
        db      0,11111111,00000000,00000000,11111111,0
        db      0,11111111,00000000,00000000,11111111,0
        db      0,11111111,11111111,11111111,11111111,0
        db      0,11111111,11111111,11111111,11111111,0
        db      0,11111111,11111111,11111111,11111111,0
        db      0,11111111,11111111,11111111,11111111,0
        db      0,11111111,11111111,11111111,11111111,0
        db      0,11111111,11111111,11111111,11111111,0
        db      0,11111111,11111111,11111111,11111111,0
        db      0,11111111,11111111,11111111,11111111,0
        .radix  10
        rept    8
        db      0,0,0,0,0,0     ;bottom blank border
        endm
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

3

```
;
; Image of a hollow diamond with a smaller diamond in the
; middle.
; There's an 8-pixel-wide blank border around all edges
; so that the image erases the old version of itself as
; it's moved and redrawn.
;
Diamond label   byte
        dw      48,6    ;height in pixels, width in bytes
        rept    8
        db      0,0,0,0,0,0     ;top blank border
        endm
        .radix  2
        db      0,00000000,00000001,10000000,00000000,0
        db      0,00000000,00000011,11000000,00000000,0
        db      0,00000000,00000111,11100000,00000000,0
        db      0,00000000,00001111,11110000,00000000,0
        db      0,00000000,00011111,11111000,00000000,0
        db      0,00000000,00111110,01111100,00000000,0
        db      0,00000000,01111100,00111110,00000000,0
        db      0,00000000,11111000,00011111,00000000,0
        db      0,00000001,11110000,00001111,10000000,0
        db      0,00000011,11100000,00000111,11000000,0
        db      0,00000111,11000000,00000011,11100000,0
        db      0,00001111,10000001,10000001,11110000,0
        db      0,00011111,00000011,11000000,11111000,0
        db      0,00111110,00000111,11100000,01111100,0
        db      0,01111100,00001111,11110000,00111110,0
        db      0,11111000,00011111,11111000,00011111,0
        db      0,11111000,00011111,11111000,00011111,0
        db      0,01111100,00001111,11110000,00111110,0
        db      0,00111110,00000111,11100000,01111100,0
        db      0,00011111,00000011,11000000,11111000,0
        db      0,00001111,10000001,10000001,11110000,0
        db      0,00000111,11000000,00000011,11100000,0
        db      0,00000011,11100000,00000111,11000000,0
        db      0,00000001,11110000,00001111,10000000,0
        db      0,00000000,11111000,00011111,00000000,0
        db      0,00000000,01111100,00111110,00000000,0
        db      0,00000000,00111110,01111100,00000000,0
        db      0,00000000,00011111,11111000,00000000,0
        db      0,00000000,00001111,11110000,00000000,0
        db      0,00000000,00000111,11100000,00000000,0
        db      0,00000000,00000011,11000000,00000000,0
        db      0,00000000,00000001,10000000,00000000,0
        .radix  10
        rept    8
        db      0,0,0,0,0,0     ;bottom blank border
        endm
;
; List of objects to animate.
;
        even    ;word-align for better 286 performance
;
ObjectList      label   ObjectStructure
 ObjectStructure <1,21,Diamond,88,8,80,512,16,0,0,350,RED>
 ObjectStructure <1,15,Square,296,8,112,480,144,0,0,350,RED>
 ObjectStructure <1,23,Diamond,88,8,80,512,256,0,0,350,RED>
 ObjectStructure <1,13,Square,120,0,0,640,144,4,0,280,BLUE>
 ObjectStructure <1,11,Diamond,208,0,0,640,144,4,0,280,BLUE>
 ObjectStructure <1,8,Square,296,0,0,640,144,4,0,288,BLUE>
 ObjectStructure <1,9,Diamond,384,0,0,640,144,4,0,288,BLUE>
 ObjectStructure <1,14,Square,472,0,0,640,144,4,0,280,BLUE>
 ObjectStructure <1,8,Diamond,200,8,0,576,48,6,0,280,GREEN>
 ObjectStructure <1,8,Square,248,8,0,576,96,6,0,280,GREEN>
 ObjectStructure <1,8,Diamond,296,8,0,576,144,6,0,280,GREEN>
 ObjectStructure <1,8,Square,344,8,0,576,192,6,0,280,GREEN>
 ObjectStructure <1,8,Diamond,392,8,0,576,240,6,0,280,GREEN>
ObjectListEnd   label   ObjectStructure
;
Data    ends
;
; Macro to output a word value to a port.
```

4

```
;
OUT_WORD        macro
if WORD_OUTS_OK
        out     dx,ax
else
        out     dx,al
        inc     dx
        xchg    ah,al
        out     dx,al
        dec     dx
        xchg    ah,al
endif
        endm
;
; Macro to output a constant value to an indexed VGA
; register.
;
CONSTANT_TO_INDEXED_REGISTER    macro ADDRESS, INDEX, VALUE
        mov     dx,ADDRESS
        mov     ax,(VALUE shl 8) + INDEX
        OUT_WORD
        endm
;
Code    segment
        assume  cs:Code, ds:Data
Start   proc    near
        cld
        mov     ax,Data
        mov     ds,ax
;
; Set 640x350 16-color mode.
;
        mov     ax,0010h        ;AH=0 means select mode
                                ;AL=10h means select
                                ; mode 10h
        int     10h             ;BIOS video interrupt
;
; Set the palette up to provide bit-plane precedence. If
; planes 0 & 1 overlap, the plane 0 color will be shown;
; if planes 1 & 2 overlap, the plane 1 color will be
; shown; and so on.
;
        mov     ax,(10h shl 8) + 2      ;AH = 10h means
                                        ; set palette
                                        ; registers fn
                                        ;AL = 2 means set
                                        ; all palette
                                        ; registers
        push    ds                      ;ES:DX points to
        pop     es                      ; the palette
        mov     dx,offset Colors        ; settings
        int     10h                     ;call the BIOS to
                                        ; set the palette
;
; Draw the static backdrop in plane 3. All the moving images
; will appear to be in front of this backdrop, since plane 3
; has the lowest precedence the way the palette is set up.
;
        CONSTANT_TO_INDEXED_REGISTER SC_INDEX, MAP_MASK, 08h
                                ;allow data to go to
                                ; plane 3 only
;
; Point ES to display memory for the rest of the program.
;
        mov     ax,VGA_SEGMENT
        mov     es,ax
;
        sub     di,di
        mov     bp,SCREEN_HEIGHT/16     ;fill in the screen
                                        ; 16 lines at a time
BackdropBlockLoop:
        call    DrawGridCross           ;draw a cross piece
        call    DrawGridVert            ;draw the rest of a
```

Code continues . . .

Abrash Code, Continued

```
        dec     bp                      ; 15-high block
        jnz     BackdropBlockLoop
        call    DrawGridCross           ;bottom line of grid
;
; Start animating!
;
AnimationLoop:
        mov     bx,offset ObjectList    ;point to the first
                                        ; object in the list
;
; For each object, see if it's time to move and draw that
; object.
;
ObjectLoop:
;
; See if it's time to move this object.
;
        dec     [bx+Delay]              ;count down delay
        jnz     DoNextObject            ;still delaying-don't move
        mov     ax,[bx+BaseDelay]
        mov     [bx+Delay],ax           ;reset delay for next time
;
; Select the plane that this object will be drawn in.
;
        mov     dx,SC_INDEX
        mov     ah,[bx+PlaneSelect]
        mov     al,MAP_MASK
        OUT_WORD
;
; Advance the X coordinate, reversing direction if either
; of the X margins has been reached.
;
        mov     cx,[bx+XCoord]          ;current X location
        cmp     cx,[bx+XLeftLimit]      ;at left limit?
        ja      CheckXRightLimit        ;no
        neg     [bx+XInc]               ;yes-reverse
CheckXRightLimit:
        cmp     cx,[bx+XRightLimit]     ;at right limit?
        jb      SetNewX                 ;no
        neg     [bx+XInc]               ;yes-reverse
SetNewX:
        add     cx,[bx+XInc]            ;move the X coord
        mov     [bx+XCoord],cx          ; & save it
;
; Advance the Y coordinate, reversing direction if either
; of the Y margins has been reached.
;
        mov     dx,[bx+YCoord]          ;current Y location
        cmp     dx,[bx+YTopLimit]       ;at top limit?
        ja      CheckYBottomLimit       ;no
        neg     [bx+YInc]               ;yes-reverse
CheckYBottomLimit:
        cmp     dx,[bx+YBottomLimit]    ;at bottom limit?
        jb      SetNewY                 ;no
        neg     [bx+YInc]               ;yes-reverse
SetNewY:
        add     dx,[bx+YInc]            ;move the Y coord
        mov     [bx+YCoord],dx          ; & save it
;
; Draw at the new location. Because of the plane select
; above, only one plane will be affected.
;
        mov     si,[bx+Image]           ;point to the
                                        ; object's image
                                        ; info
        call    DrawObject
;
; Point to the next object in the list until we run out of
; objects.
;
DoNextObject:
        add     bx,size ObjectStructure
```


Number 80 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

6

```
        cmp     bx,offset ObjectListEnd
        jb      ObjectLoop
;
; Delay as specified to slow things down.
;
if SLOWDOWN
        mov     cx,SLOWDOWN
DelayLoop:
        loop    DelayLoop
endif
;
; If a key's been pressed, we're done, otherwise animate
; again.
;
CheckKey:
        mov     ah,1
        int     16h             ;is a key waiting?
        jz      AnimationLoop   ;no
        sub     ah,ah
        int     16h             ;yes-clear the key & done
;
; Back to text mode.
;
        mov     ax,0003h        ;AL=03h means select
                                ; mode 03h
        int     10h
;
; Back to DOS.
;
        mov     ah,4ch          ;DOS terminate function
        int     21h             ;done
;
Start   endp
;
; Draws a single grid cross-element at the display memory
; location pointed to by ES:DI. 1 horizontal line is drawn
; across the screen.
;
; Input: ES:DI points to the address at which to draw
;
; Output: ES:DI points to the address following the
;               line drawn
;
; Registers altered: AX, CX, DI
;
DrawGridCross   proc    near
        mov     ax,0ffffh       ;draw a solid line
        mov     cx,SCREEN_WIDTH/2-1
        rep     stosw           ;draw all but the rightmost
                                ; edge
        mov     ax,0080h
        stosw                   ;draw the right edge of the
                                ; grid
        ret
DrawGridCross   endp
;
; Draws the non-cross part of the grid at the display memory
; location pointed to by ES:DI. 15 scan lines are filled.
;
; Input: ES:DI points to the address at which to draw
;
; Output: ES:DI points to the address following the
;               part of the grid drawn
;
; Registers altered: AX, CX, DX, DI
;
DrawGridVert    proc    near
        mov     ax,0080h        ;pattern for a vertical line
        mov     dx,15           ;draw 15 scan lines (all of
                                ; a grid block except the
                                ; solid cross line)
BackdropRowLoop:
        mov     cx,SCREEN_WIDTH/2
```

7

```
        rep     stosw           ;draw this scan line's bit
                                ; of all the vertical lines
                                ; on the screen
        dec     dx
        jnz     BackdropRowLoop
        ret
DrawGridVert    endp
;
; Draw the specified image at the specified location.
; Images are drawn on byte boundaries horizontally, pixel
; boundaries vertically.
; The Map Mask register must already have been set to enable
; access to the desired plane.
;
; Input:
;       CX - X coordinate of upper left corner
;       DX - Y coordinate of upper left corner
;       DS:SI - pointer to draw info for image
;       ES - display memory segment
;
; Output: none
;
; Registers altered: AX, CX, DX, SI, DI, BP
;
DrawObject      proc    near
        mov     ax,SCREEN_WIDTH
        mul     dx      ;calculate the start offset in
                        ; display memory of the row the
                        ; image will be drawn at
        shr     cx,1
        shr     cx,1
        shr     cx,1    ;divide the X coordinate in pixels
                        ; by 8 to get the X coordinate in
                        ; bytes
        add     ax,cx   ;destination offset in display
                        ; memory for the image
        mov     di,ax   ;point ES:DI to the address to
                        ; which the image will be copied
                        ; in display memory
        lodsw
        mov     dx,ax   ;# of lines in the image
        lodsw           ;# of bytes across the image
        mov     bp,SCREEN_WIDTH
        sub     bp,ax   ;# of bytes to add to the display
                        ; memory offset after copying a line
                        ; of the image to display memory in
                        ; order to point to the address
                        ; where the next line of the image
                        ; will go in display memory
DrawLoop:
        mov     cx,ax   ;width of the image
        rep     movsb   ;copy the next line of the image
                        ; into display memory
        add     di,bp   ;point to the address at which the
                        ; next line will go in display
                        ; memory
        dec     dx      ;count down the lines of the image
        jnz     DrawLoop
        ret
DrawObject      endp
;
Code    ends
        end     Start
```


1989 Programmer's Journal 7.2
PRODUCED 2021 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Undocumented DOS Internals

by
Raymond J. Michels

MS-DOS has been in existence for nearly a decade and is still a viable operating system for the 80X86 CPU family. If and when OS/2 becomes a fully usable product, there will be many MS-DOS users quite happy to stay where they are in the DOS world. This article will focus on some little-known or not-well-understood functions and structures of MS-DOS. This information can be very useful to the programmer try to squeeze the most out of MS-DOS. Also, I consider it a learning experience in OS architecture just to examine how MS-DOS works! In future articles I plan to target specific features with useful examples (such as a multitasking TSR to allow background tasks).

Please keep in mind that much of this information is undocumented and may not be present in all implementations of MS-DOS. ***Be careful:*** *The structures and functions described are at the heart of MS-DOS, and you can easily trash your system if you are not careful.*

Four standard operating system units will be examined:

- Process control
- Memory management
- File system
- Device management

Let's move on now to our tour through MS-DOS internals. First stop, the PSP.

Process Control

A *process* is a unique job or task that the operating system has loaded and run. Contained within a process is an area describing the job (*process control*) and the actual code and data used by the program. In MS-DOS, process control revolves around a *program segment prefix (PSP)*, which contains pertinent information related to the program at hand. Numerous processes can be loaded, but only one has control at any given time. (Multiple processes are present when memory-resident programs or TSRs have been loaded.)

The PSP is a 256-byte block of data that is built every time a program is executed. The segment address of the PSP is known as a *process ID (PID)*. (I'll describe functions to get a PID for your program later.) The basic structure of the PSP goes back to the CP/M days (ancient history!), but much has been added. The following is a MASM structure describing the PSP:

```
psp   struc
      exit_call       dw    ?
      block_length    dw    ?
                      db    ?
      cpm_call        db    5 dup (?)
      exit_routine    dd    ?
      ctrl_c_routine  dd    ?
      fatal_error     dd    ?
      parent_id       dw    ?
      handle_table    db    20 dup (?)
      environ_ptr     dw    ?
      user_stack      dd    ?
      handle_cnt      dw    ?
      handle_address  dd    ?
                      db    24 dup (?)
      dos_call        db    3 dup (?)
                      db    10 dup (?)
      default_fcb     db    16 dup (?)
      second_fcb      db    16 dup (?)
                      db    4 dup (?)
      cmd_line        db    128 dup (?)
PSP   ends
```

Much of the PSP is documented in the *DOS Technical Reference Guide* and any good MS-DOS guide, so I am going to touch upon only undocumented features.

Beginning at offset 16H, there are two components that are undocumented. The first contains the process ID of the *parent process*. The parent process is the task that has executed your program, such as COMMAND.COM or another applica-

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

tion program. This parent will still be resident in memory. The second is a table of 20 bytes. This is the *file handle table* (also known as the *job file table*). Until MS-DOS 3.3, programs were limited to opening only twenty files handles (five of which were used by standard files). This table can be relocated to a larger area to gain more file handles.

You can effect this move in DOS 3.3 through a DOS function call. In previous versions, you must modify the PSP fields that contain the number of file handles and the address of the table. One drawback of relocating the PSP file handle table: Any child processes spawned by your process will not inherit the new table.

A component called the environment pointer (see the next section) starts at offset 2CH. Starting at offset 32H are two components that deal with the file table. The first contains the maximum number of files that can be opened, which is the file handle table size. The second is a DWORD (double-word) pointer to the file handle table. (We see how the file handle table works later.)

The PSP is very important to your program. It contains information used in controlling your file access and also plays a role in memory allocation. Let's move on now to a short discussion of what is in the environment section that is connected to the PSP.

The Environment Section

The *environment pointer* is the segment address of the program's environment section. The *environment* is a data area that can be used by the associated program. Generally, data is placed in the environment by the DOS SET command, but programs can also alter this data area. While the environment section has been documented by MS-DOS, I feel it needs better explanation.

The environment is made up of a series of ASCIIZ strings [strings terminate by a null (binary 0) byte], usually in the format of **name = data**, where "name" is a variable identifier and "data" is what has been assigned by the set command. The last ASCIIZ string is followed by a second null byte to signify the end of the environment data. In MS-DOS versions 3 and later, this is followed by a word called a *byte count*; it is not clear what this word is for.

Next is an ASCIIZ string that contains the full path name of the current application. A good use of environment variables is to allow program parameters such as data paths to be specified in the environment. When COMMAND.COM executes a program, it makes a copy of its current environment and transfers this data to the new program. At program termination, this environment space is returned to free memory (unless it is a TSR). Because of this copying of the environment, programs that modify their own environments have no effect on the overall master environment of COMMAND.COM.

TSR Use of the Process ID

As stated earlier, the address of the PSP is known as the process ID. This ID is used when opening files and also for allocating memory. The MS-DOS functions 50H (**Set Current PID**) and 62H (**Get Current PID**) are important in programming TSRs. Undocumented MS-DOS function 51H also returns the PSP address in BX. However, in DOS 2.X, a switch to a DOS internal stack occurs. This stack can be damaged if another DOS function is interrupted.

Each process in MS-DOS is assigned its own file handle set. When a TSR uses files without adjusting the current PID, it actually uses the file handle table of the process in the foreground. This is bound to cause problems, since neither program knows about the other. This situation also applies to memory allocation. A TSR that allocates and deallocates memory would really be dealing with the foreground's memory blocks. By utilizing the DOS PID functions, a TSR can regulate which ID has control.

The following is a basic process that should be used by any TSR that plans to open and close files or allocate and deallocate memory:

```
TSR Initialization
     Get and save current PID (the id of the TSR)
     Initialize and stay resident
TSR Activation
     Get and save current PID (the id of the
                               foreground)
     Set PID to be the TSR (saved during
                               initialization)
     Perform TSR function
     Set PID to be the foreground process
     Return to foreground process
```

Now let's move from the PSP to MS-DOS memory allocation structures.

Memory Allocation

Every program in memory has the opportunity to allocate, deallocate, and change the size of memory blocks. The means to perform this are documented MS-DOS functions (48H, 49H, 4AH), but what is not documented is the physical structure of these memory blocks.

Each memory block created by MS-DOS has a 10H byte (one-paragraph) *memory control block (MCB)* preceding it. To find the MCB for a specific segment, subtract 1 from the segment address.

The following is a MASM data structure for an MCB. These control blocks are linked together to form a chain of MCBs.

```
mcb  struc
          Signature db   0
          Owner     dw   0     ; Segment of owner -
                               ; Usually a PID
```

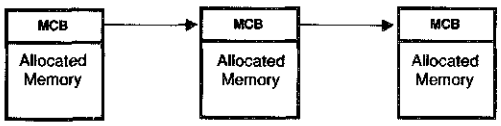

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
        Size        dw   0       ; In paragraphs
mcb  ends
```

The byte called "signature" can be either 4DH or 5AH. A 4DH signifies that this is an MCB in the chain, and the 5AH signifies the last block in the chain.

The word called "owner" specifies which segment owns this block. This is generally the owner's PSP segment address (PID). By using the owner field, DOS can make sure that one program cannot alter the size of another program's memory blocks. This protection applies *only* when using DOS memory allocation and deallocation functions.

"Size" specifies how many paragraphs are contained in this block. By adding the size value to the segment address of the MCB and then incrementing the result by 1, you can access the next block in the chain (unless the current signature is 5AH). The following figure shows a chain of memory control blocks:



Finding Our Memory

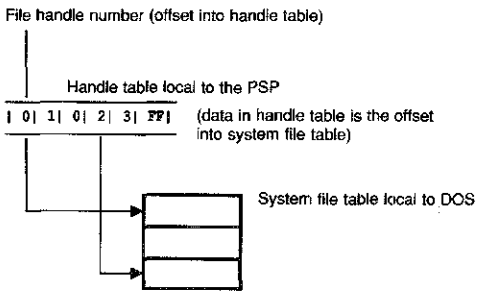
Though there is no method for going backward in the MCB chain, there does exist an undocumented way to get to the start of the MCB chain. DOS function 52H returns a pointer to a number of special DOS variables. One of these variables is a pointer to the first MCB. By using this value, we can start at the beginning of the MCB chain and trace through all allocated memory blocks.

There are numerous public domain program that use this principle to print a list of all memory-resident programs. They simply start at the beginning of the MCB chain and, each time an MCB for a PSP is encountered, display information about the PSP. A PSP is usually identified by starting with the bytes CDH and 20H (INT 20H).

MS-DOS File Management

The original MS-DOS file system is an extension of the CP/M *file control block (FCB)* system. An FCB is a block of data that contains specific information about the file (filename, file size, record position, and so on). Traditionally, this data was kept local to the program that opened the file. In DOS versions 2 and later, a UNIX-like file system called "handles" was added, giving files a more logical—rather than physical—access. The application only has to deal with a file handle number while the operating system keeps track of the specifics of the file; for example, whenever a program terminates (either normally or in error), DOS automatically closes all file handles associated with that program. It also allows an application program more flexibility in dealing with files, such as opening the same file twice to maintain separate file pointers.

When a file is opened using file handles, the handle number is returned to the application. This number is an index into the file handle table that normally resides in the PSP. (Recall that this table can be relocated to gain more file handles.) Using the offset into the file handle table, a value can be retrieved that is the offset into the MS-DOS system file table (see the next section). Later, we will look at a DOS function called **Get DOS Variables** that contains a pointer to the system file table. The following illustration describes a file handle access:


File handle number (offset into handle table)
Handle table local to the PSP
| 0| 1| 0| 2| 3| FF|
(data in handle table is the offset into system file table)
System file table local to DOS

One feature of file handles that can be used by programmers is the opening of the following series of standard files when a program is loaded:

0 - Standard Input
1 - Standard Output
2 - Standard Error Output
3 - Standard Auxiliary I/O
4 - Standard Printer Output

By writing data to file handle 1 with the DOS function 40H (file output), you will be writing to the standard output device. Offset 1 of the file handle table contains the system file table number that corresponds to the standard output device (the CRT). If I/O redirection is specified, COMMAND.COM replaces the SFT number with one that corresponds to the specified file. You can also modify the file handle table under application control to redirect output to a different device.

The System File Table

The *system file table (SFT)* is a data structure maintained in DOS that contains information about every open file in the system. It is made up of a series of blocks, with each block containing a header followed by an array of open file control blocks.

The header contains a pointer to the next SFT block and a count of the entries in the current SFT block. The following is a MASM structure for the SFT header:

```
SFT_HEAD  Struc
     SFT_Next  dd   ?
     SFT_Count dw   ?
SFT_HEAD  Ends
```

The actual SFT entry contains many fields that will be familiar if you have worked with standard file control blocks. For each SFT entry, DOS tracks the PID of the process that was opened or created and the number of processes actually accessing the file. When an open- or create-file request is issued, DOS will check the SFT entries to see if any entries already exist for that file. If so, the open count is incremented; if not, a new entry is created. The following is a MASM structure that defines the system file table:

```
SFT  Struc
     open_cnt       dw   ?    ;count of opens
     open_mode      dw   ?    ;file open mode
     Attribute      db   ?    ;Directory Attribute
     unknown1       dw   ?
```

Continued on page 36

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Programmer'sTools

from The Software Engineering Store & The Software Family

C

Borland Turbo C v2 ..... New! $99
Borland Turbo C v2 Prof. .....New! $169
Blaise C Tools Plus/5.0 ..... $99
Creative Vitamin C $169
Essential Graphics v2 ..... $232
Gimpel PC Lint $99
Ithaca HOOPS $455
Object Oriented Graphics Database
Microsoft C v5.1 ..... $299
Microsoft Quick C $68
Novell Btrieve $199
Softfocus Btree/ISAM $99
comprehesive file/index manager with all source
Wordcraft C Workshop $69
C training manual & software
Zortech C++ $99
Optimizing C++ Compiler

Turbo Pascal

Borland Turbo Pascal v5 ..... New! $99
Borland Turbo Pascal Prof. ... New! $169
Turbo Power Professional v5 .. New! $99

Other Languages & Tools

Digitalk Smalltalk/286 ..... New! $164
Borland Assembler & Debug New! $99
Matrix Layout $129
Microsoft Macro Assembler $105
Microsoft Windows SDK v2.1 ..... $328
Mortice Kern MKS Toolkit v2.3 ..... $145
KORN shell & 130 UNIX™ -like utilities
Periscope I 512K v4 .....New! $640
Periscope III v4 ..... $1125
Qualitas 386-Max $69
80386 memory manager
Solution Systems Brief ..... $call
Whitewater ACTOR $449
High productivity development for MS Windows

Free Catalog & Newsletter

We carry hundreds of products from dozens of publishers. Call or write for your free subscription to our catalog and newsletter.

Returns

We want you to pay for only the tools you really use. If you order anything featured in this ad (or most of the products in our catalog) you can open and use the product for up to 30 days and still return it if it doesn't meet your needs.

Prices subject to change without notice - ©1988 TSF All Rights Reserved

Multi-Tasking

DESQview *by Quarterdeck* is the only multi-tasking environment that provides complete compatibility for the entire PC hardware family while fully exploiting the the features of the 80286 and 80386. **$99**
QEMM **$55**
API Reference **$54**
API Ref & C Library **$179**

Disk Duplication

Duplicator Toolkit *by Copy Technologies* simplifies and speeds diskette duplication. One step operation to format, copy & verify diskette and print label — faster than just copying with DOS. **$79**

The Software Engineering Store

1-800-443-7176

In Calif: 1-800-635-0969

(415)957-0111

Visa • MasterCard
COD • Check

- Most orders shipped within 24 hours
- Free shipping on orders over $100

575 Mission Street, #600-PJ
San Francisco, CA 94105

The Software Family has a new name!

We've changed our name, but not our ownership, staff or outstanding service. The Software Engineering Store continues our tradition of fair pricing, expert technical support and honest business practices.

386 Hardware

Hauppauge 386XT replaces your XT motherboard to provide top 80386 performance with maximum compatibility. 16Mhz CPU, 1MB zero wait state memory. Provides 99.5% compatibility with both XT and AT hardware and software, and backed by a 30 day money-back compatibility guarantee. Made in U.S.A. One year warranty. **$1495**

Debugging

New version! **Periscope** *v4 by Periscope*: The Periscope debuggers provide the combinations of hardware and software you need to solve really tough programming problems. Periscope includes all the debugging features you need, but with minimum overhead and maximum flexibility. With Periscope you can debug programs which need all memory, have real-time requirements and/or are picky about vector usage.

Periscope I w/512K **$640**
0k memory overhead!
Periscope III **$1125**
hardware breakpoints and trace-back for real-time applications

Technical Support

The Software Engineering Store carefully evaluates features, quality and support before offering a product to our customers. All of the products featured in this ad are backed by excellent technical support from the publisher — in many cases directly from the program developers. All are published by companies with reasonable update policies and with a track record for responding to customer requests for both bug fixes and new features. We further back the products with our own technical support. We are willing and able to intercede with the publisher if your needs aren't satisfied or to research problems caused by the interaction of products we recommend. We can also provide quotes for custom modification of source-licensed products, in-depth technical consultation or custom-developed products.

Number 98 on the Reader Service Card.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
DCB             dd   ?    ;Address of Device
                          ;Header for Char
                          ;Devices. Address of
                          ;Drive Control
                          ;Block for Files opened
                          ;Block
                          ;Devices
Cluster_init    dw   ?    ;Initial Cluster
HHMMS           dw   ?    ;File Time
YYMMDD          dw   ?    ;File Date
File_Size       dd   ?    ;File Size
File_Pos        dd   ?    ;Current File Position
Cluster_Beg     dw   ?    ;number of clusters from
                          ;beginning
Cluster_Cur     dw   ?    ;current cluster
Block           dw   ?    ;Block Number
unknown2        db   ?    ;Directory Index
File_Name       db   8 dup (?) ; File Name
File_Ext        db   3 dup (?) ; File Extension
unknown3        db   6 dup (?)
PSP_Owner       dw   ?         ;PID that initially
                               ;opened file
unknown4        dw   ?
SFT Ends
```

Now that we have files, it would be nice to have a place to store the data. Read on.

MS-DOS Device Management

Another feature that MS-DOS borrowed from the UNIX operating system is installable device drivers. This allows for devices to be added easily at boot time without a total reconfiguring of the operating system. Both character devices (printers, COM ports, etc.) and block devices (hard disks, RAM disks, etc.) are supported and operate about the same way. Developing a driver is well documented in many publications.

Following the MS-DOS variable table is the device header of the first device driver in memory, the NUL device. From this header we can follow the chain of device drivers. (For more information on device drivers, see "DOS Device Drivers—A Debugging and Information Utility" in *PJ* 6.6.) The following is the MASM structure of the device header:

```
Device_Head    STRUC

    DVH_Next   dd   0ffffffffh     ;    Pointer to
                                   ;    next header
    DVH_Attr   dw   ?              ;    Attribute of
                                   ;    Driver
    DVH_Stat   dw   ?              ;    Driver Strategy
                                   ;    Routine
    DVH_Int    dw   ?              ;    Drive Inter-
                                   ;    rupt Routine
    DVH_Name   db   8 dup(?)       ;    Driver Name

Device_Head      ENDS;
```

Maintaining a Block Device

MS-DOS needs to know information about a block device to manage it properly. This information is stored in a structure called a *drive parameter block (DPB)*. The following is a MASM structure that defines the DPB used through DOS 3.X:

```
dpb struc

logical_num     db   ?    ;logical drive number (A=0)
drive_num       db   ?    ;drive unit number
sector_size     dw   ?    ;sector size
spa             db   ?    ;sectors per allocation unit
                          ;(cluster)
shift           db   ?    ;shift factor (2^shift)-1 =
                          ;spa
reserved        dw   ?    ;number of reserved sectors
                          ;(boot sector(s))
number_fats     db   ?    ;number of fat tables on disk
number_dirs     dw   ?    ;number of directory entries
data_start      dw   ?    ;sector where data starts
alu             dw   ?    ;number of allocation units
fat_size        db   ?    ;number of sectors in each
                          ;fat table
dir_start       dw   ?    ;sector where directory starts
device_drv      dd   ?    ;pointer to associated device
                          ;driver
media           db   ?    ;media descriptor byte
dirty_flag      db   ?    ;flags whether buffers are
                          ;'dirty'
next_dpb        dd   ?    ;pointer to next DBP
dpb_unknown     dw   ?

dpb ends
```

The DPB can be used when a program needs specific technical information about a drive, such as the number of directory entries. Utilities such as Mace and Norton use this information when accessing the disk drive directly. The address of the DPB can be acquired in two ways:

- The MS-DOS variable table contains a pointer to the first DPB.
- Undocumented functions 1FH and 32H will return a pointer to the DPB table. Function 1FH returns information on the currently logged disk drive, while 32H allows you to specify a drive.

Simply call DOS using these functions, and a pointer to the DPB will be returned in DS:BX. For function 32H, place the drive number in DL (0 = default, 1 = A, etc.).

In a future article I will describe a program that utilizes the information in the DPB to scan for bad sectors and reports what files are contained in the bad sectors.

Miscellaneous Undocumented Functions

The following functions are those that I feel a programmer will use the most. There are a number of other "reserved" functions, but they are either dummy functions to retain CP/M compatibility or are little used.

Get DOS Variables (Function 52H) In previous sections I described a table called the MS-DOS variable table. Well, here it is. Function 52H returns a pointer to this table in ES:BX. The following MASM structure represents the DOS variables for version 3.X:

```
VARS STRUC

    dpb_ptr    dd   0    ;bx+0      ;pointer to first
                                    ;DPB
    sft_ptr    dd   0    ;bx+4      ;pointer to first
                                    ;system file
                                    ;table
    clock_ptr  dd   0    ;bx+8      ;pointer to clock
                                    ;device
    con_ptr    dd   0    ;bx+0ch    ;pointer to con-
                                    ;sole device
    max_sec    dw   0    ;bx+10h    ;max sector size
                                    ;for block
                                    ;devices
    cache      dd   0    ;bx+12h    ;Pointer to CACHE
                                    ;head buffer
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

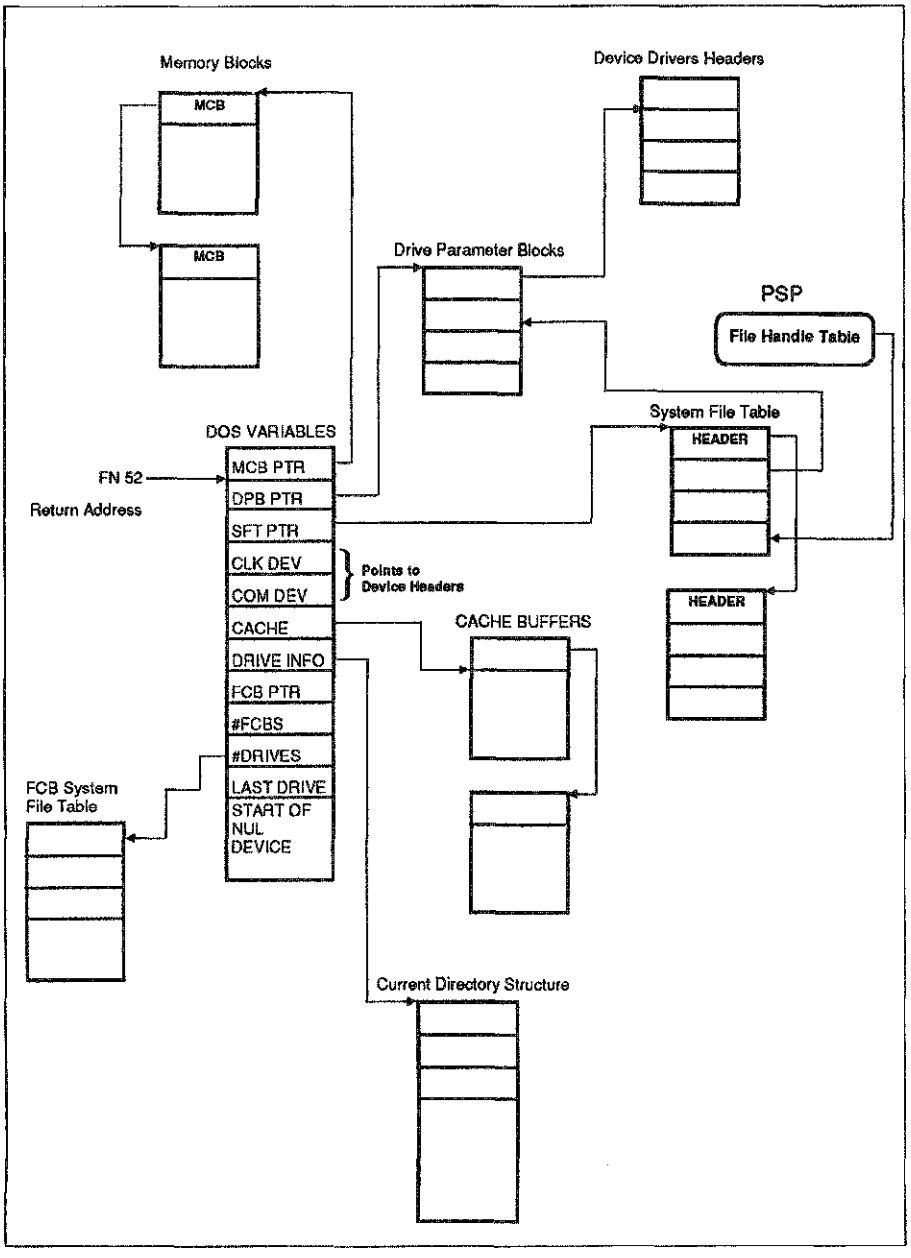
```
drive_inf dd   0    ;bx+16h    ;Pointer to cur-
                               ;rent directory
                               ;list for all
                               ;drives
fcb_ptr   dd   0    ;bx+1ah    ;Pointer to FCB
                               ;system file
                               ;table
fcb_sz    dw   0    ;bx+1eh    ;size of FCB table
drive_cnt db   0    ;bx+20h    ;number of drives
last_driv db   0    ;bx+21h    ;last drive

VARS ENDS
```

A useful piece of information that is not a part of this table is a pointer to the first memory control block segment. This pointer is this table address minus 2 bytes (the pointer is the segment address, the offset is 0).

The diagram below shows the various relationships between the MS-DOS structures that I have discussed.

Get INDOS Flag (Function 34H) DOS function 34H is very important in TSR programming. It returns a pointer in ES:BX to a byte called the MS-DOS busy flag. When this byte is nonzero it indicates that an MS-DOS function call is in progress. Since MS-DOS is not reentrant, this flag provides a means to wait until DOS can safely be called. This function should be called during TSR initialization, with the pointer saved in a local data variable for later access. A sister element of DOS is called the **DOS Safe (*or* Idle) Interrupt** (#28H):

DOS Safe Interrupt (Interrupt 28H) This interrupt is invoked when DOS is sitting idle, such as while it is waiting for keyboard entry at the COMMAND.COM prompt. This will enable a TSR to take advantage of the system's not doing anything other than waiting. During this interrupt, the DOS busy flag will be 1 since DOS is waiting, but it is safe to use DOS functions 0CH and above. A good example of the use of this interrupt is allowing a TSR that performs background disk I/O on the timer tick a chance to "steal" some extra processing time while the system is sitting idle.



Now that You Know

Well, we have just taken a whirlwind tour through the heart of MS-DOS. What can we do with this stuff, you might ask? We can debug device drivers, write trickier TSRs, and maybe even come up with ideas to supercharge MS-DOS with some multitasking ability. Knowing some of the internal workings of DOS may help spark an idea for a new application or revamping an old one.

Ray Michels is a Senior Programmer/Project Manager for Computer Entry Systems in Maryland. He also does free-lance programming and is the author of "Undocumented MS-DOS Functions," chapter 6 in The Waite Group's MS-DOS Papers. *You can write to Ray at 21 Old Farm Lane, New Freedom, PA 17349.* ♦

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

386 Now

by
M. Steven Baker

This is the first installment of a new column we've added to Programmer's Journal. *Its purpose is to illustrate 80386 code that can be used in the real mode under MS-DOS. The title was thoughtfully suggested by Jeff Duntemann, the former editor of* Turbo Technix, *whom we gratefully acknowledge. I have written this first installment, but future technical notes are up for grabs. Any takers?*

In previous articles I described a number of 80836 compilers and development tools available under DOS, using 386 DOS extenders. These 386 DOS extenders—available from Phar Lap Software, A.I. Architects, and I.G.C.—make the full power of the 386 available on the PC. Of course, 386 versions of UNIX and XENIX also bring this power to the desktop. Although I am a strong advocate of using the full power of the 80836 CPU (and not waiting for OS/386), it appears that DOS will be with us for some time. This column is an attempt to show places where the power of the 386 can be used under DOS.

This project started for me last March, when I decided to port an application over to OS/2. I chose the SEE editor that is bundled with DeSmet DC88. (The source is available at modest cost.) My first task was to convert the DeSmet C and ASM code so that it was Microsoft compatible and to get it running under DOS. The SEE editor is virtual (no file size limits), and the source code uses a **_move** routine for shifting large blocks around in memory. SEE pages text in and out and moves blocks around in 4K chunks. During the process I thought to myself that the 80386 **rep movsd** (move double words of four bytes) would be an attractive way to speed up all these memory block moves on a 386 by a factor of two. I didn't want to keep two DOS versions of the editor; so I built in a routine that patched **_move** on the fly if you were running on a 386. For this first installment, I decided to provide a replacement for a memory movement routine. Rather than use the DeSmet **_move**, I have chosen the ANSI C **memmove** routine.

Before we implement the actual **memmove** function, we need a routine to recognize when our code is running on an 80386. The assembly language procedure **cputype** should recognize a 386 in both real and virtual 8086 mode (see the listing accompanying this article). This code is adapted from the Compaq 386 technical reference manuals, but it's ultimately based on Intel guidelines. This routine works because bits 12-15 of the flag word are handled differently on the 8086, 80286, and 80386. The NEC V30 chips emulate the 8086 handling of these bits. I checked with several 386 DOS extender vendors, and they use the same technique; so it should be pretty solid. Since **pushf** and **popf** are privileged instructions, this won't normally work under OS/2 (you can actually create a comparable routine using an IOPL segment).

Based on a year-old draft of ANSI C, the prototype for **memmove** is

```
void * memmove(void *dest,
const void *src, size_t n)
```

and returns the destination pointer. The **memmove** function copies *n* characters from the object pointed to by *dest* into the object pointed to by *src* (ANSI C calls these pointers *s1* and *s2*). Copying between objects that overlap will take place correctly. ANSI C has another function, **memcpy**, for which copying overlapped objects is undefined; this function would have been much easier to implement. Most of the code in **memmove** deals with overlapping objects, moves not multiples of four, and setting up registers for the inevitable **rep movs** instruction that does the bulk of the work.

It's amazing how the simplest little project can get complicated and take more time than anticipated. I decided to see how various compiler vendors implemented this routine in their libraries. I actually have the library source code for Turbo C 1.5 and Microsoft C 5.0, but I wanted to use the most current library routines. Also, the macros in the vendor-supplied routines often obscure the functionality of the routines. So I created a simple test program using the function **memmove** compiled and linked with the /map option so that I could find the code using debug. (I actually found SOURCER much more convenient than debug for generating source code for these functions from the executables—see review in "Miscellaneous"). I looked at Microsoft C 5.1, Turbo C 2.0, DeSmet C 3.1, and MetaWare 1.4. I thought that their implementations would all be very similar. Needless to say, I was more than surprised. The example code I have supplied is based on an amalgam of the best pieces of each. Microsoft saves SI and DI (potential register variables) into BX and DX for speed, while Turbo C pushes them on the stack. The Turbo C **memmove** function calls another function, **movmem**, to do the actual work—for a further penalty in setup time. For some crazy reason, Microsoft uses **movsw** for non-overlapping blocks but **movsb** (at half the speed) for blocks that overlap. DeSmet had a minor bug but the cleanest implementation. MetaWare did not implement **memmove**, but it supports **memcpy** with in-line code generation with virtually no setup at all—clearly the fastest way.

If you cut off all the setup and overlapping object code tests, the **memmove** function basically does a **rep movs** instruction. The **rep movsb**, **movsw**, and **movsd** instructions take the same time to execute on the 80386 (5 + 4*repeat count). With the 80386, both 16-bit and 32-bit forms of many instructions are supported. Both real mode and virtual 8086 mode default to 16-bit default sizes compatible with the 8086. Two prefixes, the operand size (opcode 66H) and the effective address size (opcode 67H), allow overriding the default selection. These prefixes can precede any opcode bytes and affect only the instructions they precede. The presence of either of these prefixes toggles the default data size or effective address size for the following instruction bits. In real or virtual 8086 mode, preceding a **rep movsw** instruction with the operand size prefix toggles the operand data size to 32 bits, and voila!—we have **rep movsd**.

The sample code provided has been set up for the small model. To use with other memory models will require changing the setup and exit code slightly. I purposely didn't use macros so that the example listings would be more understandable. Implementing a fast **memcpy** function would only require stripping out all the overlapping block test stuff. The patch code is a bit complicated. I could have set a global variable for 80386 equals true that is tested on entry to the **memmove** function and then forks to a 386 version. In this case, I chose an 80386 **memmove** version for default, and the patch code moves some exit code back over the double word code to handle an 8086. Make sure you call the **cpupatch** function before you use **memmove**. Anyway, you now have a fast replacement for your compiler's **memmove** routine that works equally well (actually better than the Turbo C and Microsoft C library functions) on both the 386 and older machines.

Code follows. ♦

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

All code in this issue is available on 360K, MS-DOS disk.
ORDER BY PHONE — VISA/MC orders call toll free 800/234-0386.
Mail order information on pages 8 and 88.

Code from "386 Now" by M. Steven Baker

1

```
; CPUTYPE: This routine can be used from the real or virtual 8086 mode only.
; Otherwise will likely cause a protection violation (PUSHF instruction is privileged)
;
; Based on COMPAQ DESKPRO 386/20 Technical Reference Guide
;
; Entry:        none
; Exit:         AX = CPU type
;                    0086h if 8088/8086
;                    0286h if 80286
;                    0386h if 80386
;
        .MODEL small
        .CODE

        public          _cputype, _cpupatch, _memmove

_cputype        proc near
        pushf                   ;save the real flags register
        pop     ax              ;get register off stack
        push    ax              ;and save it again
        and     ax,0fffh        ;zero out bits 12-15
        push    ax
        popf                    ;try to put into flags
        pushf
        pop     ax              ;let's see what came out of flags
        and     ax,0F000h       ;mask off bits 12-15
        cmp     ax,0F000h       ;were these bits all 1's
        je      is_86           ;if so, it's an 8086
        pop     ax              ;get flags register to AX
        push    ax              ;and save it again
        or      ax,0F000h       ;now try to set bits 12-15
        push    ax
        popf                    ;of the flags register
        pushf
        pop     ax              ;and see what came out
        and     ax,0F000h       ;are high bits set
        je      is_286          ;if so, we have a 386
is_386: mov     ax,0386h
        jmp     short   cpu_exit
is_286: mov     ax,0286h
        jmp     short   cpu_exit
is_86:  mov     ax,86h
cpu_exit:
        popf                    ;restore flags
        ret
_cputype        endp

; void * memmove(void *dest,const void *src, size_t count)

_memmove        proc near
        push    bp
        mov     bp,sp           ;use BP to get variables
        mov     dx,di           ;save register variables
        mov     bx,si           ;into BX and DX
        mov     ax,ds
        mov     es,ax           ;force ES to DS (small model)
        mov     di,[bp+4]       ;get destination ptr
        mov     si,[bp+6]       ;and source ptr
        mov     cx,[bp+8]       ;and byte count
        mov     ax,di           ;memmove returns dest ptr
;
        jcxz    mmov_ret        ;if count=0, we're done
        cmp     di,si           ;do we have possible overlap and srcdest
        jb      forward         ;if not go do it
        je      mmov_ret        ;if dest=src, then do nothing
        mov     ax,si           ;
        add     ax,cx           ;find end point of source
        cmp     di,ax           ;do we overlap with dest ?
        mov     ax,di           ;memmove must return dest ptr
        jae     forward         ;no overlap
;
```

2

```
        add     si,cx           ;copy from end of string back
        add     di,cx
        std                     ;set reverse direction
        dec     si              ;point to last byte
        dec     di
        shr     cx,1            ;convert to words
        jnc     back2           ;carry if odd count
        movsb
back2:  dec     si              ;set index back for word move
        dec     di
patch_o1:
        shr     cx,1            ;convert to double words
        jnc     back4
        movsw
back4:  sub     di,2            ;adjust ptr back for 4 byte move
        sub     si,2
        db      66h             ;operand size prefix
patch_i1:
        rep     movsw
        cld                     ;force forward direction
mmov_ret:
        mov     si,bx           ;restore register variables
        mov     di,dx
        pop     bp
        ret
forward:
        shr     cx,1            ;byte count to words
        jnc     forwd4          ;if no carry, then even count
        movsb
patch_o2:
forwd4: shr     cx,1            ;word count to dwords
        db      66h             ;operand size prefix
        rep     movsw           ;(rep movsd) move double words
        adc     cx,cx           ;if carry was odd
        rep     movsw           ;if CX0 movsw
;
        mov     si,bx           ;restore register variables
        mov     di,dx
        pop     bp
        ret
_memmove        endp

; void cpupatch()

_cpupatch       proc near
        call    _cputype
        cmp     ax,0386h
        jne     patch_mmove
        ret
patch_mmove:
        push    ds              ;save registers
        push    es
        push    si
        push    di
        push    cx
        mov     ax,cs           ;set DS=ES to CS
        mov     ds,ax
        mov     es,ax
        mov     si,offset patch_i1      ;input patch
        mov     di,offset patch_o1
        mov     cx,(offset forward- offset patch_i1)
        rep     movsb
;
        mov     si,offset patch_i1
        mov     di,offset patch_o2
        mov     cx,(offset forward - offset patch_i1)
        rep     movsb
;
        pop     cx
        pop     di
        pop     si
        pop     es
        pop     ds
        ret
_cpupatch       endp

end
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

OS/2 Dynamic Link Libraries

by
Ray Duncan

Copyright © 1989 Ray Duncan
Adapted from Advanced OS/2 Programming, *Microsoft Press, 1989*

OS/2 dynamic link libraries (also called dynlink libraries or DLLs) are essentially subroutine packages that are bound to an application at load time or during execution, rather than at link time. But such a terse definition conveys little of the generality and power of the dynlink library concept:

- Code and read-only data in dynlink libraries is shared invisibly between all of the processes ("clients") that use the libraries, conserving physical memory and disk swap space.
- Code and data in dynlink libraries can be loaded on demand; they do not occupy physical memory until they are needed.
- Dynlink library routines can be referenced symbolically (that is, by name); so, a library and the applications that use it can be independently maintained and improved.
- Centralization of frequently used routines in dynlink libraries reduces the size of each application program file and conserves disk space.
- Dynlink libraries can contain I/O privilege (IOPL) segments that access the hardware directly. DLLs can therefore function much like device drivers for noninterrupting devices, but they provide a more flexible interface to applications than the simple open, read, and write calls supported by the kernel's file system.

Dynlink libraries were pioneered and proven in real-mode Windows, an environment that makes the most of a severely limited address space with a complex software emulation of virtual memory. But in OS/2, which can rely on hardware support for virtual memory and memory protection, dynlink libraries truly come into their own. Dynlink library segments reside in the global memory pool and can be freely moved, swapped, discarded, and reloaded on the basis of overall system activity.

In fact, OS/2's dynlink library support can be thought of as a general-purpose overlay facility that is fully integrated with the system's linker, loader, and virtual memory manager. Much of OS/2 *itself*—including most Application Program Interface (API) entry points and the code for API functions that do not need to run in kernel mode—is distributed in the form of dynlink libraries. This rather elegant unification of OS/2's services and its implementation minimizes its fixed memory requirements.

The routines in a dynlink library execute in user mode in the context of the client process. In other words, as far as the OS/2 kernel's scheduler is concerned, there is no difference between a thread executing within a process and within a dynlink library bound to the process, and no stack switching is involved; the library is viewed as an extension of the process. Consequently, the routines in a dynlink library have unrestricted access to the calling process's memory and such other resources as file, pipe, semaphore, and queue handles.

From an application's point of view, however, every dynlink library (regardless of its origin) is indistinguishable from a component of the operating system. Each DLL routine accepts its parameters on the stack, is entered by a **far** call to a named entry point, and returns a status code in AX. The details of how a DLL routine does its work are invisible to the application; the routine may continue to execute in user mode within the DLL, invoke an IOPL segment, request action by the kernel on the application's behalf, or use interprocess communication mechanisms to request action by a server process on the same or another machine.

The system loader has yet another perspective. To the loader, every executable file, whether it contains a DLL or an application, is just another system resource called a *module*. (The term as it is used here should not be confused with object modules, source code modules, or others.) While a program is running or a dynlink library is in use, the segment and entry point tables from the file header are kept resident so that discardable segments and load-on-demand (LOADONCALL) segments can be easily located when they are needed.

If an additional copy of the same program is started, or if another process references the same library, the loader already has the necessary information in hand; so, the system overhead to create the new process or dynamic link is relatively small. The loader maintains a use count for each module. When all the instances of a program have terminated, or all the clients of a dynlink library have released it or terminated, the module use count becomes zero, and the loader discards the module and associated tables and reclaims the memory.

This behavior of the loader has an interesting side effect: Any time that a program is running or a dynlink library is in use, the file containing that module is held open by the loader with a sharing mode of *deny write;* during this time, the file cannot be renamed, erased, or replaced. (This improves performance and reliability by eliminating multiple open/close operations and ensuring that a discardable segment cannot be modified between loads.) Since the system's base set of DLLs is always in use by the Presentation Manager shell, you can't

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

replace one of these DLLs on your fixed disk unless you reboot the system from a self-contained floppy disk that does not reference them.

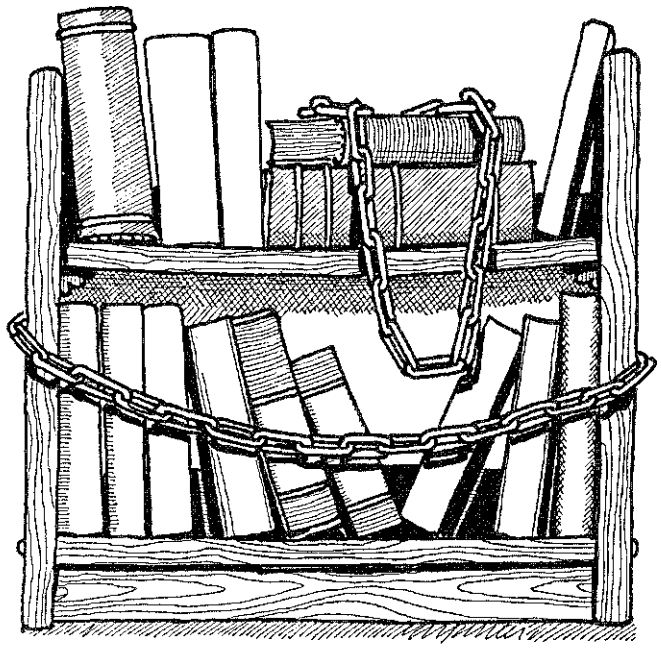
Dynlink libraries differ in important ways from the two other kinds of libraries found in OS/2: object libraries and import libraries. Dynlink libraries are structured just like executable program (EXE) files except that they have the extension DLL and a flag is set in the file header so that they cannot be run directly from the command line. The routines in a DLL are suitable for immediate execution and are bound *(dynamically linked)* to an application at its load time or run time.

In contrast, object libraries and import libraries are used at link time rather than at execution time and are in a special format understood by the LINK and LIB utilities. The code in an object library requires further processing by the linker before it can be executed, and it is permanently incorporated *(statically linked)* into a program or a DLL's executable file. The records in import libraries do not (usually) contain executable code at all; they are merely stubs representing individual DLL routines, and they signal the linker to build the information necessary for dynamic linking into a program's EXE file header.

Dynamic Linking at Load Time

Dynamic linking at load time is easy to arrange. In a MASM program, the routines to be dynlinked are simply declared as external with the **far** attribute; for example:

```
extrn DosWrite:far
```



In a C program, the routines are declared as external with the **far pascal** (parameters pushed left to right, called routine clears stack) attributes; for example:

```
extern unsigned far pascal DosWrite( ... ) ;
```

If you #include the API header files supplied with OS/2 development tools, these external declarations are made for you automatically.

Then, when you are linking your program, you either specify the module and entry point names for the dynlinked routines with IMPORTS statements in the application's module definition (DEF) file, or you link the program with an import library that matches the dynlink library or libraries you are going to use at run time. For example, the import library OS2.LIB or DOSCALLS.LIB contains import records for the OS/2 kernel's API entry points. The result of either method is the same: the linker builds the names of the dynlink libraries and routines into tables in the EXE file header, along with pointers to each reference to the dynlink libraries in the executable code.

When your program is loaded for execution, the system loader inspects the file header's *module reference table* to determine which dynlink libraries will be used by the process. If these libraries are not already loaded, the loader invokes itself recursively to bring them (and any libraries they reference in turn) into memory. The loader always assumes that a dynlink library has the extension DLL and can be found in one of the directories named in the LIBPATH directive in CONFIG.SYS. If any dynlink library cannot be located, or if memory is exhausted during the loading of libraries, the client program will not be loaded.

Some dynlink libraries have an initialization routine that must be called when they are loaded for the first time *(global initialization)* or each time they are bound to a new process *(instance initialization)*. The type of initialization required, if any, is specified in the library's file header, as is the entry point of the initialization routine. The initialization routine may return an error code if it can't locate resources that the dynlink library will need later (such as a font file, device, or additional memory), causing loading of the client program to be aborted.

When all necessary libraries have been loaded and any necessary initialization of those libraries has been completed, the loader inspects the segment tables for each module and creates a *local descriptor table (LDT)* selector for each program and dynlink library segment. All segments that are marked PRELOAD are brought into memory, and any necessary fixups are performed. (Segments marked LOADONCALL are not brought into memory and

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

DosFreeModule
Release linkage between process and dynlink library

DosGetModHandle
Return handle for dynlink library if it is already loaded

DosGetModName
Retrieve fully qualified filename of dynlink library

DosGetProcAddr
Obtain address of entry point within dynlink library

DosLoadModule
Load dynlink library if not already loaded, and return handle for use with **DosGetProcAddr**

Figure 1. Summary of OS/2 API calls used by applications to link to DLLs at runtime

fixed up until they are referenced during program execution.)

Imported dynlink library routines can also be specified in a module definition file by an *ordinal* rather than by name. An ordinal is the index to the routine's position in the library's entry point table. Using ordinals speeds up dynamic linking and reduces the size of your application's EXE file (since the imported name table in the header will be smaller). But linking with ordinals has its disadvantage: Either the ordinals must not be allowed to change from one version of the library to another, or the library and all the applications that use it must be kept synchronized. Either way, use of ordinals forfeits much flexibility.

Dynamic Linking at Run Time

Instead of allowing the linker and system loader to do all the work of dynamic linking automatically, you can elect to have your program create dynamic links to the libraries it needs while it is executing, on a function-by-function basis. The API functions for run-time dynamic linking are summarized in Figure 1.

To establish dynamic links at run time, the program must first ask the system loader to map the dynlink library's segments into its own memory space by calling **DosLoadModule**. If the library is found, any libraries that it references are also loaded (as described earlier for load-time dynamic linking) and a module handle is returned. If the library (or a library that it uses) is not found, the name of the missing library is returned in the calling process's buffer, along with an error code.

Assuming that **DosLoadModule** is successful and a handle is obtained, the program must then obtain the entry points for each of the library functions that it will use. The function **DosGetProcAddr** is called with a module handle and the address of an ASCIIZ entry point name or ordinal. It returns a selector and offset for the entry point in a variable. The library routine can then be invoked by pushing the appropriate parameters onto the stack and performing an indirect **far** call through the variable.

After the program is finished using the dynlink library, it can release it by calling **DosFreeModule** with the module handle. If no other process is using the library, it will be discarded and the memory it occupied will be reclaimed. Once **DosFreeModule** has been called, any entry points that were previously obtained for the library with **DosGetProcAddr** are no longer valid, and any attempt to use the library will cause a general protection fault. Listing 1 contains a skeleton example of run-time dynamic linking in a MASM program.

The two other API functions for dynamic linking at run time are rarely used. **DosGetModName** accepts a module handle (for a library or a process) and retrieves the fully qualified filename for the module. **DosGetModHandle** accepts an ASCIIZ module name and returns a handle for the module if it is already loaded or an error if it is not loaded; **DosGetModHandle** is strictly an existence test and does not increment the module's use count.

Designing New Dynamic Link Libraries

Designing and writing a new dynlink library is a straightforward process in either MASM or C. During the planning stage, there are three design issues you must address: the DLL interface to applications, DLL initialization, and DLL data segments.

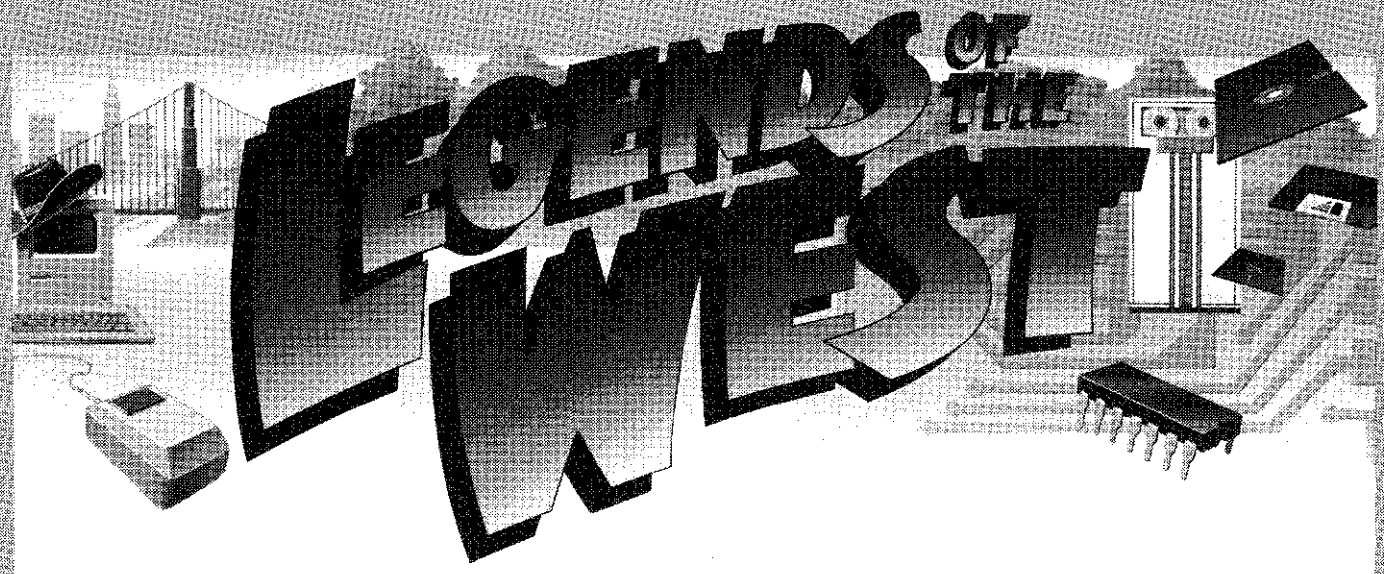
The range and level of complexity for the functions that the DLL will export to applications must be gauged carefully. If the functions are too few and too high level, the application programmer will be forced to write his or her own routines for special cases. If the routines are too numerous, the programmer will be hard put to remember what is available. And if the routines are too primitive, the application programmer will not gain enough leverage by using the functions to make the effort worthwhile. The **Vio** subsystem that is supplied with OS/2 is an excellent example to emulate.

When you are writing a MASM DLL for use with MASM applications, the parameter passing between the application and the DLL can be register based, stack based, or a hybrid of the two. When you are writing a DLL in C (or to be used with C applications) you don't have quite so much freedom: The structure of the C language more or less forces you to use the same stack-based API as the "standard" OS/2 DLLs, leaving open only the question of what each "function" will return as its "value." In either case, however, I strongly recommend that you follow the kernel API conventions and pass all parameters to the DLL on the stack, returning a status code in AX and any other information to the caller's variables. Symmetry with the rest of the system will pay off as fewer exceptions to remember and, hence, fewer bugs to fix.

An initialization routine in your DLL is optional. If one is present, its entry point is specified by an END statement in a MASM source file. DLLs written in C that require initialization must, therefore, have at least one small module written in MASM, though the MASM routine which receives control at initialization time may call a C function to do the actual work.

If the DLL has an initialization routine, you must determine whether it will be called only when the library is first loaded (global initialization) or each time the library is attached by a client process (instance initialization). Global initialization is especially appropriate when a DLL controls a device: If the device is not available, the initialization routine returns an error. Instance initialization is useful when the DLL will allocate private data structures or system resources on a per-client basis. In such cases, the initialization routine should

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



The West. The place of dreams, imagination and legend. The Gold Rush. The 1906 earthquake. Silicon Valley. The legendary West Coast Computer Faire, birthplace of the personal computer revolution.

For 13 years the West Coast Computer Faire has been responsible for the introduction of more new products that have become standards, more dissemination of industry information and more products seen, bought and used than any other public computer show. An event where the search for the leading edge technologies of today and the news of technologies to come, requires only a walk down the aisles.

This year the West Coast Computer Faire features a major concentration on exhibits and conference sessions in the areas of:

- Desktop Publishing (including computer graphics for business and entertainment)
- Local Area Networking (LANs)
- PC CAD
- Macintosh/IBM connectivity
- HyperMedia
- Vertical Markets
- Programming Tools and more.

Come to the Faire if you are looking for the latest in hardware, software, peripherals and support programs — and create a legend in your own time. The West Coast Computer Faire, where the legend continues March 17–19, 1989 at Brooks Hall and The Civic Auditorium, San Francisco. The best in the west.

PJ WILL BE AT THE FAIRE. DROP BY OUR BOOTH WHILE YOU ARE THERE.

THE 14TH WEST COAST COMPUTER FAIRE

SHOW HOURS: Friday – 12pm – 8pm • Saturday – 10am – 6 pm • Sunday – 10am – 4pm

Produced by The Interface Group, Inc. ©1989

TICKETS INCLUDE CONFERENCE & EXHIBITS

$20 for all 3 days • $12 for 1 day

Available at the Door • For more information call (617) 449-6600 x5028

Number 58 on the Reader Service Card

Don't Miss the COMPUTER CURRENTS QUIZATHON Win fabulous prizes!

Saturday, March 18, 1989 at the West Coast Computer Faire.

Watch for details in **Computer Currents.**

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

also register a routine for each client with **DosExitList** so that it will be notified when the client terminates and can clean up any resources or data that have been left hanging.

The question of the DLL's data segments goes hand in hand with global or instance initialization. Some DLLs may not need any data segment of their own—if they do not have any initialized data, do not need to store any information across calls from the application, and can use the stack for all their transient variables. Other DLLs, such as those controlling a single system resource such as a device, may need only a single data segment to store the state of the device and synchronize accesses to it by multiple processes.

But the more common case is the DLL that needs multiple unshared segments to store information across function calls on a per-client basis. Each time a new process attaches to the library, OS/2 loads a fresh copy of the DLL's data segments to map into the process's memory space, using identical selectors in each process's LDT so that the same relocations in the DLL's shared code segments will serve for all. The selector magic inherent in OS/2's support for dynlink instance data is one of the most subtle but exquisite features of OS/2's design. During instance initialization, a DLL can also dynamically allocate memory that it can use on behalf of the client process.

While the nature of the DLL's application interface is defined by its source code, the initialization type (global or instance) and behavior of each segment (preload or load on call, shared or unshared) is controlled by keywords in the module definition file.

Writing DLLs in MASM

Listing 2 contains a code skeleton for an assembly language DLL named MASMDLL.DLL. Procedures that serve as entry points must be declared **public** and **far** since they are entered by an intersegment call; procedures that are used only as subroutines within the DLL itself can be either **near** or **far**. Each such procedure should set up a stack frame so that it can access its parameters, save any registers that it is going to change, and reset the DS register to make its data segment addressable (if it has one). To exit, a procedure should restore the caller's registers (except for AX) and then use RET with an offset to clear the parameters (if any) from the caller's stack.

If an initialization routine is present, it is identified by the END directive and is also coded as a **far** procedure. Before calling the DLL initialization routine, OS/2 sets up the registers as follows:

Register	Contents
CS:IP	Entry point defined in END statement
SS:SP	Current user stack
DS	Selector for DLL's DGROUP, if any; otherwise, selector for DGROUP of process attaching to DLL
AX	Module handle of the DLL

All other register values are undefined. If the module saves a copy of its own handle, it can pass it to client applications later, and they can use it to retrieve the library's fully qualified filename with **DosGetModName**.

The initialization routine exits with a **far** return, passing a status code in AX. Unlike the conventions used elsewhere in OS/2, initialization routines return zero to indicate an error and a nonzero value to indicate success. If the status returned is zero, the program that referenced the library will not be loaded, and the name of the failing library will be passed to that program's parent.

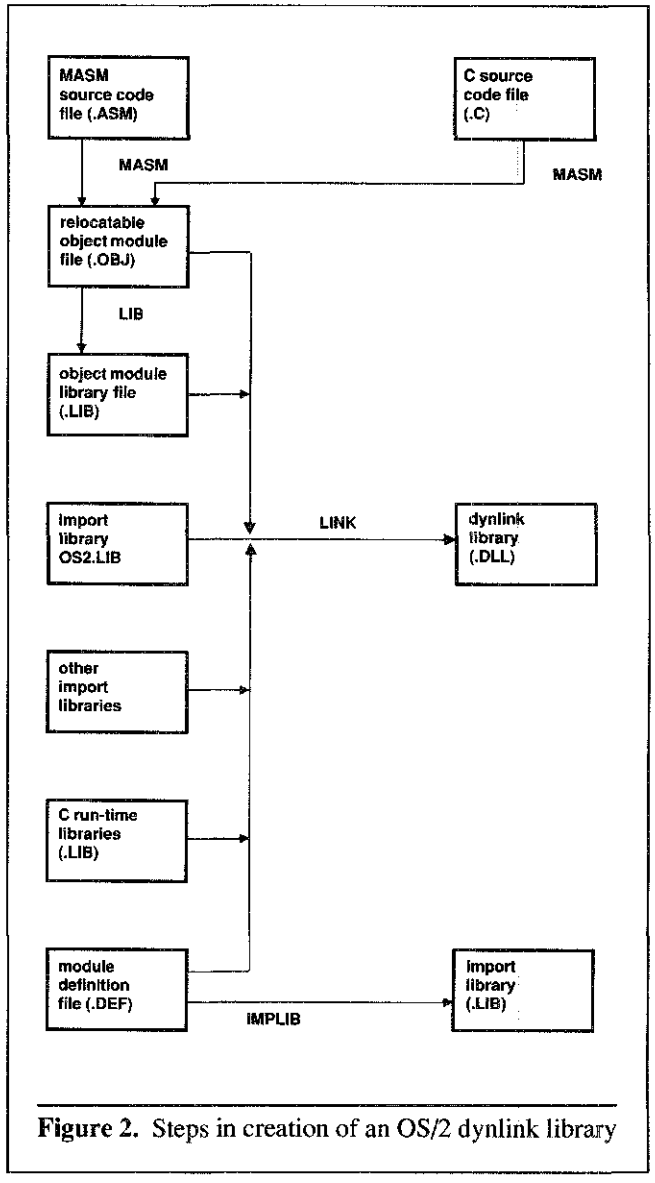


Figure 2. Steps in creation of an OS/2 dynlink library

The module definition file for our skeleton assembly language DLL is shown in Listing 3. Real DLLs with many segments may have elaborate DEF files, but this one is quite simple. The key elements are the LIBRARY directive, which signals the linker that a dynlink library is being built and that a stack declaration may be absent, and the EXPORTS statement, which designates the entry points for applications.

Instance or global initialization and DGROUP for the DLL are controlled by optional parameters to the DEF file's LIBRARY and DATA statements. PRELOAD or LOADONCALL should be assigned to each segment based on how likely it is to be used. In a simple DLL with only two segments (such as this skeleton example), both might as well be marked PRELOAD so that the time required to fetch them from disk will be hidden inside the overall load time of the application.

To build our hypothetical dynlink library, you would enter the following sequence of commands:

```
[C:\] MASM MASMDLL;
[C:\] LINK MASMDLL,MASMDLL.DLL,,OS2,MASMDLL.DEF
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Note the use of OS2.LIB, which is the import library for the OS/2 kernel API entry points. For our purposes in this article, the import libraries OS2.LIB and DOSCALLS.LIB are interchangeable.

Writing DLLs in C

If you prefer to write your DLL in C, a few special tricks are needed. First of all, each C function that will serve as an entry point from an application must be declared **far pascal**. Other functions, which are used only within the library, may use whichever style of parameter passing you prefer, although the **pascal** convention generates smaller, faster code. Close attention must be paid to every pointer, since a dynlink routine is in some ways an extreme example of mixed-model C programming. Finally, the statement

```
int _acrtused = 0
```

should be included to prevent the C startup code from being dragged into the library by the linker. When compiling your dynlink library, you must use the following switches:

/c	compile only (to allow you to link separately)
/Asnu	small model, assume DS not equal to SS, save DS and reset it to point to the named data segment at entry to each function
/Gs	disable stack checking

In some cases, you may also need to use the **/ND** switch to name the data segment that will be made addressable when a DLL function is entered.

The initialization routine, if any, must be written in MASM and assembled separately, using the **/Mx** switch so that case will be preserved in public and external names. The C-language body of the DLL must then be linked together with the MASM startup routine, the OS/2 kernel import library, and an appropriate module definition file.

Listing 4 contains the skeleton source code for a simple C dynlink library named CDLL.C. Listing 5 (CINIT.ASM) contains the assembly language source code for the initialization entry point, and Listing 6 is the module definition file (CDLL.DEF). In order to build the dynlink library CDLL.DLL, you would enter the following commands:

```
[C:\] CL /c /Asnu /Gs CDLL.C
[C:\] MASM /Mx CINIT;
[C:\] LINK /NOI /NOD CDLL+CINIT,CDLL.DLL,,OS2,CDLL.DEF
```

If you use C run-time library functions in your DLL, you must be sure to use the special reentrant libraries provided for multithreaded applications so that your DLL can be used safely by more than one process or subprocess (thread) at a time.

Building Import Libraries

If your dynlink library contains many entry points, you can make life much easier for the programmers who use it by creating a matching import library. As we already noted, an import library can be searched by the linker to satisfy references to dynlink routines, eliminating the need for specific IMPORTS statements in the application's module definition file.

Import libraries are built by the IMPLIB.EXE utility, which is supplied in the OS/2 Programmer's Toolkit. The parameters

Parallel Programming for "C"

INTERWORK™
A Concurrent Programming Toolkit

Interwork is a "C" program library which allows you to write your programs as a set of cooperating concurrent tasks. Very useful for simulation, real-time applications, and experimentation with parallel programming.

FEATURES

- Supports a very large number of tasks (typically more than 100) limited only by available memory. Low overhead per task results in very fast context switching.
- Provides a full set of inter-task communication (ITC) facilities, including shared memory, locks, semaphores, blocking queues, and UNIX™-style signals. Also has building blocks for constructing your own ITC facilities.
- Handles interrupts (DOS version) and integrates them into task scheduling. Supply your own interrupt handlers or block tasks on interrupts.
- Lets you trace task switches and inter-task communication.
- Comes with complete documentation including a user's manual and reference manual of commands.

Interwork is available for the following systems:

Hardware	*Operating System*	*Price*
IBM PC, XT, AT	PC-DOS 2.0 or later	$129
IBM PC AT	XENIX™	$159
DEC VAX™, SUN	UNIX 4.2BSD	$249

PC-DOS version is compatible with DeSmet, Lattice, and Microsoft C compilers.

Please specify hardware and operating system when ordering. Shipping and handling included; COD orders add $2.50. Send check or money order to:

Block Island Technologies
Innovative Computer Software
13563 NW Cornell Road, Suite 230, Portland, Oregon 97229-5892
(503) 241-8971

Trademarks: Interwork, Block Island Technologies; UNIX, AT&T Bell Laboratories, Inc.; XENIX, Microsoft, Inc.; VAX, Digital Equipment Corporation

Number **244** on the Reader Service Card

OS/2 USERS
End Floppy Boots!
with
MultiBoot:
The First OS/2 Booting Utility.

MultiBoot

- **Boots DOS or OS/2 *without* floppies.**
- **Automatically boots your preferred system.**

Order MultiBoot if you:

- **Evaluate or Support OS/2 and DOS.**
- **Work in OS/2 and DOS.**
- **Want easy access to both operating systems.**

MultiBoot eliminates the need for floppy disks to boot your OS/2 or DOS operating system and gives you the option to select your choice of operating system.

MultiBoot is designed for OS/2 compatible IBM PC AT, PS/2 and true compatibles. Requires DOS version 3.0 and up. Installs easily.

INTRODUCTORY OFFER

Order MultiBoot For Only
$49.95* + $3.00 shipping and handling.
Send check or money order to:
MultiBoot, Bolt Systems, Inc. 4340 East-West Highway
Bethesda, Maryland 20814,
or call **1-301-656-7133, FAX: 301-907-8736,**
and order by Visa/MasterCard.
Specify 3.5" or 5.25" diskette. * Maryland residents add 5% sales tax.

A Product of BOLT Systems, Inc.

Number **265** on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

required by IMPLIB are the name of the import library to be built and the name of one or more library definition files containing EXPORTS statements for dynlink library entry points. The command line for IMPLIB takes the form

```
IMPLIB libfile.LIB deffile.DEF
            [ deffile.DEF ... ]
```

where the extensions LIB and DEF must be supplied as shown.

For example, to create an import library for our skeleton assembly language example MASMDLL.DLL, the command line would be

```
[C:\] IMPLIB MASMDLL.LIB
          MASMDLL.DEF
```

Once MASMDLL.LIB is available, we can remove the IMPORTS statements from the module definition files of any application programs that reference MASMDLL.DLL and link those applications against the import library instead; for example,

```
[C:\] LINK MYAPP,,,OS2+MASMDLL,
          MYAPP.DEF  <Enter>
```

The entire process of building a dynlink library and import library from MASM or C source code (or both) is diagrammed in Figure 2.

Ray Duncan, author of the MS-DOS "bible," Advanced MS-DOS, *and general editor of the* MS-DOS Encyclopedia *(both published by Microsoft Press), is president of Laboratory Microsystems Inc. LMI sells Forth interpreters, compilers, and cross-compilers for a variety of microprocessors and does consulting work in assembler, Forth, and C. Ray's latest book,* Advanced OS/2 Programming, *was published in February.*

Code follows. ♦

All code in this issue is available on 360K, MS-DOS disk.
ORDER BY PHONE — VISA/MC orders call toll free 800/234-0386.
Mail order information on pages 8 and 88.

Code from "Dynlink Libraries" by Ray Duncan — 1

Listing 1. Example code sequence for runtime dynamic linking.

```
objbuf  db      64 dup (0)      ; receives failing module
                                ; or entry point name

objbuf_len equ  $-objbuf

mname   db      'VIOCALLS',0    ; module name

mhandle dw      0               ; receives module handle

ename   db      'VIOGETANSI',0  ; entry point name

eptr    dd      0               ; receives far pointer
                                ; to entry point
        .
        .
        .
                                ; attach to DLL...
        push    ds              ; receives failing dynlink
        push    offset DGROUP:objbuf
        push    objbuf_len      ; length of buffer
        push    ds              ; address of module name
        push    offset DGROUP:mname
        push    ds              ; receives module handle
        push    offset DGROUP:mhandle
        call    DosLoadModule   ; transfer to OS/2
        or      ax,ax           ; module found?
        jnz     error           ; jump if no such module

                                ; get entry point...
        push    mhandle         ; module handle
        push    ds              ; address of entry name
        push    offset DGROUP:ename
        push    ds              ; receives entry pointer
        push    offset DGROUP:eptr
        call    DosGetProcAddr  ; transfer to OS/2
        or      ax,ax           ; entry point found?
        jnz     error           ; jump if no entry point
        .
        .                       ; other processing here...
        .
        call    eptr            ; indirect call to
                                ; dynlink library routine
        .
        .                       ; other processing here...
        .
                                ; module no longer
                                ; needed, release it...
        push    mhandle         ; module handle
        call    DosFreeModule   ; transfer to OS/2
        or      ax,ax           ; should never fail
        jnz     error           ; but check anyway
        .
        .
        .
```

Listing 2. MASMDLL.ASM, the source code for an assembly language dynlink library named MASMDLL.DLL.

```
DGROUP  group   _DATA

_DATA   segment word public 'DATA'
        .
        .
        .
_DATA   ends

_TEXT   segment word public 'CODE'

        assume  cs:_TEXT,ds:DGROUP

        public  MYFUNC
MYFUNC  proc    far             ; dynlink routine

        push    bp              ; set up stack frame
        mov     bp,sp
        push    ax              ; save affected registers
        push    bx
        push    cx
        push    dx
        push    si
        push    di
        push    ds

        mov     ax,DGROUP       ; make DLL's data
        mov     ds,ax           ; segment addressable
        .
        .
        .
        pop     ds              ; restore registers
        pop     di
        pop     si
        pop     dx
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
        pop     cx
        pop     bx
        pop     ax
        pop     bp

        ret     pars*2          ; clear stack, return
                                ; to caller
MYFUNC  endp

INIT    proc    far             ; initialization routine
        .
        .
        .
        ret

INIT    endp

_TEXT   ends

        end     INIT            ; initialization entry point
```

Listing 3. MASMDLL.DEF, the module definition file for the skeleton MASMDLL.DLL.

```
LIBRARY MASMDLL INITGLOBAL
CODE PRELOAD
DATA PRELOAD MULTIPLE
EXPORTS
  MYFUNC
```

Listing 4. CDLL.C, a skeleton dynlink library written in C. The function MYFUNC is exported for dynamic linking by application programs. The initialization function C_INIT is called when the DLL is first loaded from the entry point in CINIT.ASM. Note that unlike other OS/2 functions, a DLL initialization function must return a nonzero value to indicate success.

```
int _acrtused = 0;          /* don't link C startup */
      .
      .
      .
/*
    MYFUNC is exported for dynamic linking
    by application programs.
*/

int far pascal MYFUNC(int a, int b)
{
      .
      .
      .
}
      .
      .
      .
/*
    C_INIT is called from the entry point in CINIT.ASM
    when a client process dynlinks to the library.
*/

int far pascal C_INIT(void)
{
      .
      .
      .
    return(1);              /* return success code */
}
```

Listing 5. CINIT.ASM, the MASM module containing the initialization entry point for CDLL.DLL. The MASM routine INIT simply calls the C function C_INIT to do the actual work of initialization.

```
        extrn   C_INIT:far

_TEXT   segment word public 'CODE'

        assume  cs:_TEXT

INIT    proc    far             ; initialization routine

        call    C_INIT          ; call C routine to
                                ; do the actual work

        ret                     ; back to caller with
                                ; AX = status code
                                ; ("value" from C_INIT)
INIT    endp

_TEXT   ends

        end     INIT
```

Listing 6. CDLL.DEF, the module definition file for CDLL.DLL.

```
LIBRARY CDLL INITINSTANCE
PROTMODE
CODE PRELOAD
DATA PRELOAD MULTIPLE

EXPORTS
  MYFUNC
```


TRUE MULTITASKING
With
MultiDos Plus
"multitasking for the IBM-PC."

Ideal for developing applications in process control, data acquisition, communications, and other areas. Check these features which make **MultiDos Plus** an unbeatable value.

- Run up to 32 programs concurrently.
- Your software continues to run under DOS. No need to learn a new operating system.
- Use the compilers you already have. Supports software written in most languages.
- Operator commands to load/run programs, change priority, check program status, abort/suspend/resume programs.
- Programmatic interface via INT 15H for the following.
  - * Intertask message communication. Send/receive/check message present on 64 message queues.
  - * Task control by means of semaphores. Get/release/check semaphores.
  - * Change priority-256 priority levels.
  - * Suspend task for specified interval.
  - * Spawn and terminate external and internal tasks.
  - * Disable/enable multitasking.
  - * and more!
- Independent foreground/background displays.
- Access to DOS while applications are running.

Hardware/Software Requirements

IBM PC/XT/AT or true clone. Enough memory to hold **MultiDos Plus** (48 KB) and all your application programs. Also may need 4 or 16 KB memory for "hidden screens" for each active task. MS-DOS (or PC-DOS) 2.0 or later operating system.

only: $24.95 *OR* $99.95 with source code

Outside USA add $5.00 shipping and handling.

Visa and Mastercard orders only call toll-free: 1-800-872-4566, ext. 350., or send check or money order to:

NANOSOFT
13 Westfield Rd, Natick, MA 01760
MA orders add 5% sales tax.

Number 109 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

OS/2 Multitasking with Class

by
Vaughn Vernon

In my first article ("Object Oriented OS/2", *PJ* 6.5), I focused on C++, discussed the reasons that make many feel that C++ has promise, and noted some commercially available C++ translators and compilers. We didn't get much further into OS/2 than the date and time facilities, which are available on most systems and not just on OS/2.

In this issue, we will focus on the heart of OS/2: multitasking. We'll also develop a couple of useful OS/2 classes to control multitasking and put them to work.

When reading this article, you may find it helpful—maybe even necessary—to refer to *The C++ Programming Language*, by Bjarne Stroustrup. This book is the definition of C++; it can well be referred to as the K&R (Kernighan and Ritchie's *The C Programming Language*) of C++. No doubt, in attempting to cover a particular facility, I will have to leave out some details because of space constraints. Please refer to the Stroustrup book for more thorough coverage.

Cool Threads, Man

Under OS/2, tasks are performed concurrently. The OS/2 kernel uses preemptive scheduling to control the execution of multiple tasks. Actually, only one instruction is performed at any given time, but the kernel uses preemption to switch tasks on a regular basis. This provides the illusion of multiple concurrent tasks.

Because of the multitasking capabilities of OS/2, many processes can be in memory at the same time. However, unlike most multitasking operating systems, OS/2 does not schedule and execute processes; it schedules and executes *threads*. A process is an owner of resources, such as code, data, and file handles, and threads are the dispatchable entities of a process.

When a process is first spawned, it has one thread of execution. What sets OS/2 apart is its ability to manage multithreaded processes. Since multiple threads are part of a process, each thread in the same process can share the process's common resources, such as data and file handles.

A thread is executed via a C++ (or other) function. In other words, you tell OS/2 to execute a function asynchronously, under its own schedule. To tell OS/2 which function should be executed as a thread, you pass OS/2 the far address of the function. Also, each thread needs a stack from which it may create local variables and by which it passes arguments and its return address to other functions. The thread's stack is allocated dynamically from the memory-free store and initialized by the creating thread prior to thread creation. (Details are covered later in this article.)

Imagine the power of using threads in a spreadsheet application: While one thread is recalculating a spreadsheet in one window, another thread is formatting a printer page, a third thread is displaying a graphics report, and yet another thread is managing user input into a new spreadsheet window.

As you might guess, multiple threads can cause problems if they are not well behaved. For instance, the multiple threads of our spreadsheet application could corrupt data if one or more threads attempted to modify data cells as another thread tried to read them.

Software Traffic Lights

Enter OS/2 *interprocess communications (IPC) facilities*. Threads use IPC facilities to inform one another of their well-being and activities. OS/2 IPC provides such tools as semaphores, pipes, message queues, and shared memory.

Here, we are interested in semaphores, which will act as our "software traffic lights." Semaphores allow one thread to tell others, "Hey, I'm in here! Stay out until I tell you it is okay." This helps protect critical sections of computing. For example, a spreadsheet's printer thread could use a semaphore to tell the other threads not to modify the cells of its spreadsheet while it is printing.

In this case, semaphores protect resource ownership. When a thread wishes to access a serially reusable resource, the thread asks the operating system for semaphore ownership. When ownership is obtained, all other threads asking for semaphore ownership are blocked (suspended) by the operating system until the current owner relinquishes ownership. Optionally, a thread may choose not to block on the semaphore but is informed not to access the shared resource at that time. This allows the thread to continue doing something else.

And They're Off . . .

On the other hand, OS/2 threads provide another important multitasking facility: signaling. Signaling with semaphores is like a race official firing a pistol to start the race. In a signaling environment, one thread will self-set a semaphore and wait for another thread to clear it. The thread responsible for clearing the semaphore would do so only after a given event took place.

For example, a communications application could use semaphore signals to post an event such as a modem needing to be answered. The auto-answer thread would self-set the auto-answer semaphore and block on its clearance. When a polling thread found that the modem needed to be answered, it would clear the auto-answer semaphore, unblocking the

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

auto-answer thread—like firing the starting gun while the runners are on their marks.

Semaphore signaling should not be confused with signals produced by software interrupts (such as Ctrl-Break), which are completely different.

A Thread/Task Class

Since we are solving problems using an object-oriented approach, it is appropriate to introduce some new C++ classes. Listing 1 contains the code for the Task class. The header file that defines the Task class is named task.hxx:

The beginning of the file has some preprocessor directives that may look a little confusing. Don't pay any attention to them for now; instead, let's concentrate on the Task class definition. There is a hidden data item, **returnCode**, whose uses will become clear in a moment.

Notice the class constructor, **Task::Task()**. It takes two arguments: the far address of the thread function, and the size that the thread stack should be. First, the constructor dynamically allocates the thread's stack. Since the stack normally grows downward under an Intel architecture (the 80386 allows several stack types), the stack pointer is set to the high address, or the top of the stack (**stack += stackSize**). Finally, the OS/2 system function used to create a new thread, **DosCreateThread()**, is called. The thread routine address, a pointer to the thread ID variable, and the stack pointer are passed. The thread ID variable, **taskID**, could be used to control thread priorities if the Task class implemented this feature.

The Task class does nothing in its class destructor. If you are concerned about wasting memory on an undeleted thread stack, reconsider. If the class destructor were to delete the thread stack, it could, in effect, pull the rug out from under the thread. For instance, the function (and possibly the thread) that created the new thread could end (return or DosExit) and cause the Task object to go out of scope before the new thread was completed. A deleted stack segment would most certainly cause an OS/2 general protection fault (or at least memory corruption) the next time an attempt was made to access the segment.

The remaining class function, or *method* is used to check the status of the thread. (When you invoke a class method, you are said to be sending a message to the object for which you invoked the method.) The **Task::GetStatus()** method is used to see if the thread was spawned at all. If it returns zero, the thread is executing or is at least scheduled for execution. Another return value means that something has gone wrong.

An IPC Semaphore Class

Next, let's look at the Semaphore class, which is defined in semaphor.hxx. The Semaphore class works under both resource-control and signaling environments. Also, the Semaphore class supports both types of OS/2 semaphores: system and RAM. OS/2 system semaphores are created, opened, and closed just like a disk file. All system semaphores reside in OS/2 kernel filename space under a pseudodirectory named \SEM. They are not associated with the OS/2 file system other than for naming conventions. To create a system semaphore for printer resource control, you might use the name

```
"\\SEM\\PRINTER.001"
```

(Note that the "\\" creates a single "\" in C and C++.)

OS/2 RAM semaphores are not allocated by the kernel; they are allocated in application data or stack space. For instance, to create a RAM semaphore, you simply declare and initialize a long variable:

```
long aPrinterSemaphore = 0L;
```

Each different semaphore type has its distinct advantages. Proper use is determined by your particular needs. For instance, RAM semaphores are much faster to use, but a RAM semaphore is not particularly easy to use across process boundaries (by threads in more than one process). You would need to put the semaphore in shared memory, creating more work and a potential for problems.

The way the Semaphore class constructor, **Semaphore::Semaphore()**, is called determines the type of OS/2 semaphore you wish to use. If you create a Semaphore object without arguments to the class constructor, you will use a RAM semaphore:

```
Semaphore printerSem;
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Software Tools

For Programmers & Non-Programmers

★ NEW ★ VERSION **Opt-Tech Sort**™ ★ NEW ★ VERSION

Opt-Tech Sort is a high performance Sort/Merge/Select utility. It can read, sort and write a file faster than most programs can read the data. Example: 1,000 records of 80 bytes can be read sorted and a new file written in less than 9 seconds (IBM XT). Opt-Tech Sort can be used as a stand-alone program or Called as a subroutine to over 30 programming languages.

Numerous features are included. A partial list of features includes: The ability to process files of any size. Many filetypes are supported including Sequential, Random, delimited, Btrieve, dBASEII & III and many others. Up to 30 key fields can be specified over 16 different types of data are supported. Powerful record selection capability allows you to specify which records are to be included on your output. Record reformatting allows you to change the structure of your output record and to output special fields such as record numbers for use as indexes. **Newest Features: Faster than ever! LIM EMS support, User exits to pass or receive records. Less disk space required and much more!** MS-DOS $149.

On-Line Help™

On-Line Help allows you to easily add Help Windows to all your programs. On-Line Help is actually two help packages in one. You get BOTH Resident (pop-up) and Callable help systems.

The resident version allows you to add help to any system. Your Help system is activated when the "Hot Keys" that you specify are pressed. You can then chain between help windows in any manner you desire. Upon clearing the help windows you are right back into what ever program you left.

The callable version allows you to easily display help windows from your programs. A simple Call to the help system makes the window appear. On-Line Help is callable from over 20 different languages.

You have full control over the help window content, size, color and location. MS-DOS $149. Demo $10.

Scroll & Recall™

Scroll & Recall is a resident screen and keyboard enhancement. It allows you to conveniently scroll back through data that has gone off the top of your display screen. Up to 27 screens of data can be recalled or written to a disk file (great for documenting systems operations). Also allows you to easily recall and edit previously entered DOS commands without retyping.

Scroll & Recall is very easy to use. Its a resident utility that's always there when you need it. MS-DOS $69.

Visa, M/C, AMEX, Check, Money Order, COD or Purchase Orders accepted

To order or to receive additional information just call and receive immediate highly qualified attention!

Opt-Tech Data Processing

P.O. Box 678 / Zephyr Cove, NV 89448

(702) 588-3737

Number 16 on the Reader Service Card

However, if you supply a semaphore path name and ownership and creation specifications, you will use a system semaphore:

```
Semaphore globalPrinterSem("\\sem\\printer.001", 0, 1);
```

This creates a system semaphore that can be used by the creating process and by any other process that opens it. For instance, the following code shows how another process would open (BOOL create argument of 0) the previously created semaphore:

```
Semaphore myPrinterSem("\\sem\\printer.001", 0, 0);
```

If you will be using a semaphore for resource control, two class methods will be of use to you: **Semaphore::Request()** and **Semaphore::Clear()**. When you wish to gain ownership of a resource, you can call **Semaphore::Request()**:

```
myPrinterSem.Request( -1L );
```

Passing the **timeOut** argument as -1 says that you are willing to wait indefinitely for ownership. Otherwise, an argument of 0 or greater specifies how many milliseconds you are willing to wait for ownership. If a time out is used, the **Request()** method will return a TRUE value if ownership was obtained or FALSE if a time out occurred before the current owner cleared the semaphore.

After obtaining the semaphore ownership and processing in the critical section, you will need to relinquish semaphore ownership. You do this by sending the semaphore object a message to clear ownership:

```
myPrinterSem.Clear();
```

If you wish to use a semaphore in a signaling environment, other methods are available:

```
Semaphore::SetNow()
Semaphore::Wait()
Semaphore::Fire()
```

Additionally, I have implemented some C++ operator overloading for the Semaphore class. Basically, C++ allows you to create class methods that are used like standard C++ operators, such as ++. The methods may or may not take arguments of objects of the class type. For instance, if you had a class called Apple, here is how you could add the values of two Apples:

```
Apple a1, a2, a3;

 //...
a3 = a1 + a2;
```

In this case, the Apple + operator method gives the user the impression that Apples are a native data type in the C++ language. The + operator method would take a1 and a2 as arguments and return an Anple as a result. (The = operator could also be overloaded if there where more to assigning Apples than a bit copy.)

However, under the Semaphore class, I implement only the ++ and -- operators. The ++ operator is used to set a signaling semaphore, and the -- operator is used to clear the semaphore. In essence, using the -- operator fires the starting gun for the thread waiting for the event.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Debug Multi-Megabyte DOS Programs

There was a time when the only way to debug very large DOS C programs meant suffering under 640KB memory constraints. Not any longer. *Instant-C* opens a new age of debugging under DOS.

Instant-C is specifically designed to debug large DOS programs. It works in "protected mode" on all 80286 and 80386-based PCs giving you access to **16MB** of extended memory. *Instant-C*'s source-level debugging lets you debug and run programs in excess of 100,000 lines of code without worrying about DOS memory limitations or resorting to overlays. Hard to believe? It's true. No other DOS debugger can live up to *Instant-C*'s standards.

And *Instant-C*'s debugging capabilities are unsurpassed. With features such as automatic run-time checking, data monitoring, source code animation, source code backtracing, and unlimited conditional breakpoints at your disposal, debugging your large programs will be faster than ever before.

Instant-C also includes an integrated C compiler, linker, editor, and source formatter. With *Instant-C* you get a powerful integrated tool package for working with your C programs. And *Instant-C* is compatible with your Microsoft C compiler.

Save yourself hours of debugging time every single day. Try *Instant-C* and discover how to make debugging your large DOS C programs quicker than you ever believed possible. With *Instant-C*'s **30 day no risk money back guarantee** the only thing you've got to lose is bugs.

- **Debug Multi-Megabyte Programs**
- **Automatic Run-time Checking**
- **All Debugging in C**
- **Unlimited Conditional Breakpoints**
- **Compatible with your Assembly Code**
- **Source Code Backtracing**
- **Support for Full C Language**
- **Data Breakpoints**
- **Source Code Animation**
- **Compatible with Microsoft C**

Rational Systems' 30 Day No Risk Money Back Guarantee

If for any reason you decide not to keep *Instant-C*, just return it to us within 30 days of purchase and we will gladly refund your money.

Order *Instant-C* today!

***only* $795**

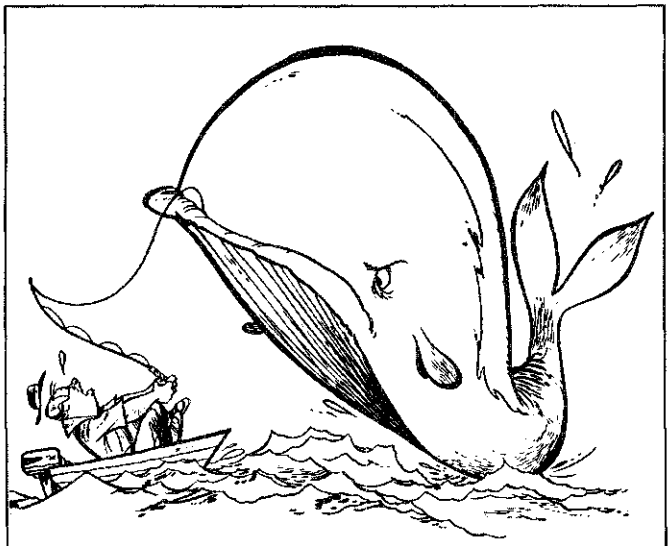
Rational
Systems, Inc.

220 North Main Street
Natick, MA 01760
Tel: **(508) 653-6006**
FAX: **(508) 655-2753**

Trademarks are acknowledged to Rational Systems, Inc., and Microsoft Corp.

Number 20 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



Is your C program too much to handle?

You need PC-lint 3.0

PC-lint will analyze your C programs (one or many modules) and uncover glitches, bugs, quirks, and inconsistencies that your compiler, working on one module at a time, will miss.

PC-lint will find uninitialized variables, function inconsistencies, argument/parameter mismatches, wild pointers, inconsistent declarations, unaccessed variables, suspicious macros, unreachable code, indentation irregularities, unusual expressions, format irregularities, and ... **New with 3.0** array and `struct` access and initialization, `case` fall-through, precision tracking, complete `const` checking and more. Support for Microsoft 5.1, Turbo 2.0, Watcom.

Customize PC-lint to your own programming style and project. Turn off any error message locally or globally, alter size of scalars, adjust format of error messages, turn off errors on the basis of symbol name and more.

Mainframe and mini programmers -- Use Generic Lint in shrouded source form on VAX/VMS, Unix, Xenix, Microport, OS-9, IBM VM/CMS - MVS, AOS/VS etc. Requires only K&R C to compile. Prices start at $798. Call for details.

(215)584-4261

Only $139 - CALL TODAY

30 Day Money-back Guarantee

Gimpel Software

3207 Hogarth Lane Collegeville PA 19426

Works with any compiler - Specify MS-DOS or OS/2
Quantity and educational discounts available.
PA orders add 6% sales tax -- Outside USA add $20
PC-lint and Generic Lint are trademarks of Gimpel Software.

Contact this advertiser directly

There are some other overloaded C++ operators for the Semaphore class, but they produce only error results when they are used. Stroustrup recommends this method for preventing a user of the class from inadvertently misusing operators with disastrous results.

Putting the Classes to Work

The final listing is the program share.cxx. This program uses the Task and Semaphore classes to demonstrate how the primary thread (the first thread on process startup) would spawn a secondary thread and how they use a semaphore to coordinate the sharing of a character buffer and the C++ ostream file. The ostream semaphore is of system type so that threads in other processes can coordinate writing to standard output without cluttering the display.

In addition to a couple of resource-control semaphores, the **startingGun** semaphore is used as a signal from the secondary thread to the primary thread that it is ready to begin processing. The rules say that when the two threads begin processing, neither thread is allowed to access aBuffer or ostream without its respective semaphore ownership.

With Style and Class

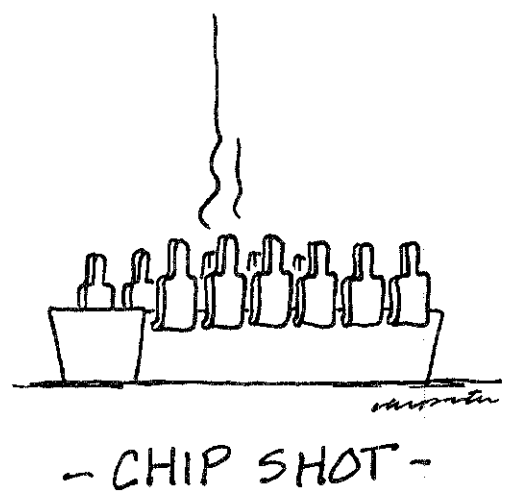
Glancing back at the Task and Semaphore definitions, you will see that their class definition files contain some bizarre-looking preprocessor directives. These are used to prevent spurious redefinition errors from occurring. These frustrating errors can be the result of using multiple classes or derived classes in one C++ program file.

Basically, each class definition should be self-contained. The user of the class should not have to include header files for the class definition—the class should handle itself, including any header files it may need. If a class is derived from another class, the derived header definition file should contain an inclusion of the parent-class header definition.

In short, that is the purpose of the preprocessor directives in the Task and Semaphore header files. Notice the Task class definition in Listing 1 again. If (for whatever reason) the Task class header has already been included, the symbolic constant **__TASKCLASS__** will have already been defined. The **#ifndef APIENTRY** and **#ifndef INCL_DOS** directives prevent the os2def.h and bsedos.h headers from being reincluded.

Vaughn Vernon is president of Aspen Scientific, producers of CURSES and FORMATION, user interface cross-development tools for UNIX, MS-DOS, and OS/2. Vaughn is co-author of The Advanced C Programmer's Guide to OS/2, *to be published by Microsoft Press in 1989.*

Code follows. ♦



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

All code in this issue is available on 360K, MS-DOS disk.
ORDER BY PHONE — VISA/MC orders call toll free 800/234-0386.
Mail order information on pages 8 and 88.

Code from "OS/2 Multitasking . . ." by Vaughn Vernon

1

```
LISTING 1
// task.hxx
//
// implements the Task class.  this manages the spawning of an OS/2 thread.
//
// author: vaughn vernon
// © Copyright 1988 Aspen Scientific
// All Rights Reserved.
# ifndef __TASKCLASS__
# define __TASKCLASS__
# ifndef APIENTRY
    include <os2def.h>
# endif

# ifndef INCL_DOS
#     define INCL_DOS
#     include <bsedos.h>
# endif
// the Task class
class Task {
    USHORT  returnCode;
public:
    // constructor, it spawns the thread
    Task( VOID auto (FAR * routine)(), USHORT stackSize ) {
            PBYTE stack;    // the stack pointer
            TID taskID;     // the thread/task ID
            stack = (PBYTE) new BYTE[ stackSize ];
            if ( stack != PBYTE(0) ) {
                // the stack grows down
                stack += stackSize;
                // spawn thread
                returnCode = DosCreateThread( routine,
                                              (PTID) &taskID, (PBYTE)stack );
            }
            else
                returnCode = TRUE;      // in this case bad
    }
    // the destructor does nothing
    // the stack is freed by OS/2 when the Task completes.
    ~Task()
        { ; }
    // methods
    // was the thread executed?
    BOOL    GetStatus()     {
        return returnCode ? FALSE:TRUE;
    }
};
#endif
LISTING 2
// semaphor.hxx
//
// defines the OS/2 semaphore class for IPC
//
// author: vaughn vernon
// © Copyright 1988 Aspen Scientific
// All Rights Reserved.
#ifndef __SEMCLASS
# define __SEMCLASS__
# ifndef APIENTRY
    include <os2def.h>
# endif
# ifndef INCL_DOS
    define  INCL_DOS
    include <bsedos.h>
# endif
// the semaphore class definition
class Semaphore {
    union   {
                ULONG   RAM;            // a RAM Semaphore
                HSYSSEM SYS;            // a System Semaphore
            } Sem;
    HSEM    SemH;                       // this actual handle
```

Code continues . . .

All 6 issues from 1988—
containing the highly acclaimed Michael Abrash
On Graphics series
plus the VGA disk
Special Limited Time Offer

only **$35.00**

A savings of **$17.00** off the regular price

To order call toll free

800-234-0386

Or see our ad on page 88

FULL AT&T C++: ANNOUNCING VERSION 1.2

Guidelines announces its port of **version 1.2** of AT&T's C++ translator. As an object-oriented language C++ includes: classes, inheritance, member functions, constructors and destructors, data hiding, and data abstraction. "Object-oriented" means that C++ code is more readable, more reliable and more reusable. And that means faster development, easier maintenance, and the ability to handle more complex projects. C++ is **Bell Labs' answer to Ada and Modula 2.** C++ will more than pay for itself in saved development time on your next project.

C++

from GUIDELINES for the IBM PC: $295

Requires IBM PC/XT/AT or compatible with 640K and a hard disk.
Note: C++ is a *translator,* and requires the use of Microsoft C 3.0 or later.

Here is what you get for $295:

- The full AT&T v1.2 C++ translator.
- *Libraries* for stream I/O and complex math.
- **The C++ Programming Language,** the definitive 327-page tutorial and description by Bjarne Stroustrup, designer of C++.
- Sample programs written in C++.
- Improved installation guide and documentation.
- 30-day money-back guarantee.

NOW AVAILABLE FOR UNIX V/386 – $495

To Order:
send check or money order to:

GUIDELINES SOFTWARE, INC.
P. O. Box 749, Dept. PJ
Orinda, CA 94563

To order with VISA or MC, phone (415) 254-9183.
(CA residents add 6% tax.)

C++ is ported to the PC by Guidelines under license from AT&T.
Call or write for a free C++ information package.

Number **204** on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

2

Vernon Code, Continued

```
USHORT  returnCode;             // a good create/open?
// a private error routine
VOID    ErrorOp( CHAR *s )
          cout << "\nclass: Semaphore: bad operator: " << s << "\n";
        }

        // the class methods

public: // constructor and destructor
        // system sem. constructor
        Semaphore( PSZ sysSemName, USHORT owner, BOOL create ) {
          if ( create )
            returnCode = DosCreateSem(
                           (owner ? CSEM_PRIVATE:CSEM_PUBLIC),
                           (PHSYSSEM) &Sem.SYS, sysSemName );

          else
            returnCode = DosOpenSem( (PHSYSSEM) &Sem.SYS,
                           sysSemName );
                           // use system handle
            SemH = Sem.SYS;
        }
        // RAM sem. constructor
        Semaphore( )                    {
            Sem.RAM = 0L;               // initialize
            SemH = &Sem.RAM;            // handle is address; use RAM handle
            returnCode = 0;
        }
        // destructor may have to close the system resource
        ~Semaphore( )                   {
            if ( Sem.SYS != HSYSSEM(0) )
                DosCloseSem( Sem.SYS );
        }
        // control methods
        // was the open/create valid?
BOOL    GetStatus()     { return (returnCode ? FALSE:TRUE); }
// SetNow is a forced set, usually meant for thread signaling
VOID    SetNow()        { DosSemSet( SemH ); }
// SetRequest is used to wait for a possibly owned semaphore
BOOL    SetRequest( LONG timeOut ) { return (DosSemRequest( SemH, timeOut ) ? FALSE:TRUE);
        }
// Wait is used to wait for the clearing of a semaphore.
// this is typically used to wait for a semaphore signal.
BOOL    Wait( LONG timeOut )    {
          return DosSemWait( SemH, timeOut );
        }
// Fire is used to send a signal to a waiting thread
// that self-set this semaphore
VOID    Fire()          { Clear(); }
// Clear is used to clear ownership of this semaphore
VOID    Clear()         { DosSemClear( SemH ); }
// C++ operator overloading:  implements a signaling
// environment. "++" sets, and "--" fires.
VOID  operator++()      { SetNow(); }
VOID  operator--()      { Fire(); }
// invalid C++ operators for this class; although this
// is not really necessary, BS recommends it for
// users of the class other than the inventor.
// for instance, I sure don't want someone trying to
// add semaphores!:  sem3 = sem1 + sem2;
VOID  operator+()       { ErrorOp( "+" ); }
VOID  operator-()       { ErrorOp( "-" ); }
VOID  operator+=()      { ErrorOp( "+=" ); }
VOID  operator-=()      { ErrorOp( "-=" ); }
VOID  operator()()      { ErrorOp( "()" ); }
VOID  operator[]()      { ErrorOp( "[]" ); }
VOID  operator*()       { ErrorOp( "*" ); }
        // etc....

};
#endif
LISTING 3
// share.cxx
//
// tests the Task and Semaphore classes
//
// author: vaughn vernon
// © Copyright 1988 Aspen Scientific
```


NEW VERSION 2

MICROSOFT, TURBO AND MIX POWER C PROGRAMMERS...
C WINDOWS TOOLKIT™ PUTS YOU IN CHARGE OF VIDEO *and* DATA-ENTRY

NEW FEATURES

- *Comprehensive data-entry and verification procedures with scanf( ) type syntax.*
- *Predefined data-entry functions for dates, SSNs, telephone numbers, currency amounts with commas, time, scientific notation, etc.*
- *Turn any window into an editor with your choice of keyboard mapping (Wordstar™ by default).*
- *Virtual windows up to available memory.*
- *Save windows to disk.*
- *Nested windows.*
- *Create and edit VGA/MCGA/EGA/Hercules fonts of any size.*

WINDOWING FUNCTIONS
- Create pop-up windows
- Create pull-down menus
- Create spreadsheet menus
- Create context-sensitive help screens
- Store windows in RAM or on disk
- Modify windows out of view
- Use 8 different types of exploding windows
- Scroll and move windows

MicroWay 386 SUPPORT*
- Runs in protected mode
- Uses 32-bit instruction set

*Extra cost option-call

SYSTEM CONTROL
- Device-independent output
- Supports VGA/MCGA/EGA/Hercules/ CGA/MDA
- Detect the types of video adapter installed
- Switch between adapters
- Detect ANSI.SYS
- Control the size and position of the cursor
- Detect the enhanced keyboard
- Disable the video signal
- Delay program execution to microsecond resolution

VGA/EGA SUPPORT
- Use all 64 VGA/EGA colors
- Use VGA 28/43/50 line modes
- Use EGA 43-line mode
- Use 2 fonts simultaneously
- Design and edit custom fonts with FONTEDIT,™ our font editor (included)
- Smooth scroll the screen
- Smooth pan the screen

FAST SCREEN I/O
- Write formatted output (like **printf( )**)
- Get snow-free output on the CGA
- Scroll the screen
- Use direct screen writes or the BIOS

HERCULES SUPPORT
- Create and edit your own Ramfonts™
- Detect the presence of a Hercules Card
- Detect Ramfont™ support
- Load and store Ramfonts™ to RAM or disk
- Switch between modes
- Detect whether the Hercules Card is in text or graphics mode

Full source code included — No run-time royalties.
Includes 230 page manual — 30-day Money-Back Guarantee

Magna Carta SOFTWARE

P.O. Box 475594
Garland, TX 75047-5594
FAX: (214) 226-0386

Requires: IBM PC, XT,AT, PS/2 or compatible.
Supports Microsoft C 5.0/5.1/Quick C, Borland Turbo C 1.5, Mix Power C

CALL (214) 226-6909

MasterCard VISA **Only $99.95**
(Texas Residents add 8% sales tax.)

Number **212** on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
// All Rights Reserved.
#include <stream.hxx>
#include <os2class/task.hxx>
#include <os2class/semaphor.hxx>

// the routine executed as a Task
VOID FAR aRoutine( VOID );
// the resource to protect
CHAR aBuffer[ 36 ];
// global RAM semaphore objects
Semaphore bufSem;
Semaphore startingGun;

int
main()
{
    // get ready for the race;
    // self set signalling semaphore
    ++startingGun;
    Semaphore streamSem( "\\sem\\ostream.cls", 1, 1 );
    // create a global global ostream semaphore
    // spawn a task; os/2 thread
    Task loop( aRoutine, 4096 );
    if ( loop.GetStatus() == FALSE ) {
        cerr << "\nError: the task was not run!\n";
        DosExit ( EXIT_PROCESS, 2 );
    }
    // wait for the task to begin
    startingGun.Wait( -1L );
    for ( int i=0; i < sizeof (aBuffer); ++i) {
        bufSem.SetRequest( -1L );
        aBuffer[i] = 'A' + i;
        bufSem.Clear();
        DosSleep(0L);
    }
    bufSem.SetRequest( -1L );
    aBuffer[ sizeof( aBuffer ) - 1 ] = char(0);
    streamSem.SetRequest( -1L );
    cout << "\nMain thread finished, buffer is:\n" << aBuffer << "\n";
    cout.flush();
    streamSem.Clear();
    bufSem.Clear();

    // wait for thread to finish
    startingGun.Wait( -1L );
    DosExit( EXIT_THREAD, 0 );
}

// the task/thread routine
VOID FAR
aRoutine()
{
    // for sharing ostream
    Semaphore streamSem( "\\sem\\ostream.cls", 1, 0 );

    // fire the startingGun for this and main thread
    --startingGun;
    ++startingGun;
    for ( int i=0; i < sizeof (aBuffer); ++i ) {
        bufSem.SetRequest( -1 );
        aBuffer[i] = 'a' + i;
        bufSem.Clear();
        DosSleep(0L);
    }
    bufSem.SetRequest( -1L );
    aBuffer[ sizeof( aBuffer ) - 1 ] = char(0);
    streamSem.SetRequest( -1L );
    cout << "\nSecondary thread finished, buffer is:\n" << aBuffer << "\n";
    cout.flush();
    streamSem.Clear();
    bufSem.Clear();

    // i'm finished!
    --startingGun;
    DosExit( EXIT_THREAD, 0 );
}
```


PJ CODE

Code disks from six issues, only $39.95*

PROGRAMMER'S JOURNAL
CODE DISKSCRIPTION—

Subscribe Today!
You'll receive your code disks in the mail every other month throughout the year. Learn from *PJ* code or use it in your own programs. Each disk contains all of the program listings published in *PJ* plus .EXE files, .OBJ files, benchmark source files, all for **only $39.95!***
If purchased separately, the regular price for these six disks is $60.00.
So save yourself some time, effort and over $20.00—order your *PJ DiskScription* today!

For faster service with VISA/MC call toll-free:
800-234-0386
* Prepaid, US only. Call or write for prices outside the US.

FINALLY!
UNLIMITED MEMORY C++ FOR MS-DOS AND UNIX

INTEK C++

- ***Compile larger programs.*** Access up to 4 GB with Intek C++ running in 80386 protected mode and get full usage of C++'s powerful object-oriented programming features.
- ***Choose the compiler you prefer.*** Microsoft C, Turbo C, Metaware, or Watcom—it's up to you.
- ***Enhance program compatibility.*** Intek C++ supports C extended keywords, so it is compatible with programs that run under Microsoft Windows or OS/2.
- ***Simplify your development environment.*** Intek C++ can cross-compile from Unix System* V/386 to DOS, so you can keep everything in one place.

For just $495, Intek offers you more.

INTEK
INTEGRATION TECHNOLOGIES
1400 - 112th Avenue SE, Suite 202
Bellevue, Washington 98004
(206) 455-9935 FAX: (206) 455-9934
Usenet: uunet!intek01!info

Requires 80386, 640K (1 MB or more recommended), C compiler. Note that 4 GB access refers to compile time; runtime access depends upon your C compiler. Metaware, Microsoft, and Watcom are all trademarks of their respective companies: Turbo C is a trademark of Borland Intl; MS-DOS is a trademark of Microsoft; Unix is a trademark of AT&T. Intek is licensed from AT&T. * Unix version available second quarter 1989.

Number 239 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

A Simple Debugging Utility for Presentation Manager and Windows

by
William S. Hall

WPMAUX is the collective name for two programs, WINAUX (for Windows) and PMAUX (for Presentation Manager). This kind of utility does not replace a true debugger; rather, it is used to trace and display actions as they occur without significantly interfering with the program under analysis. Each displays character strings received from another Windows or PM program. A typical application might be to print the contents of messages or to determine what a Windows or PM function *really* does.

In many ways, this article is a tribute to Michael Geary's OX.SYS. An amazingly useful device driver, OX.SYS allows sharing the AUX device with a secondary monochrome adapter, such as the IBM MDA or a Hercules board. It is an invaluable tool in Windows programming. One simply has to run Windows with redirection to AUX; then all statements directed to **stdout** (which is the bit bucket in Windows) appear on the monochrome screen. Of course, OX.SYS requires the proper hardware, and it has not yet been extended to OS/2. The two programs presented here help to fill this void.

One very clever feature of OX.SYS is that for each new line, the screen is first scrolled up and the next line is placed at the bottom of the screen. As a result, the display code is very simple. We have followed the same approach here. Consequently, the WPMAUX window can be placed near the bottom of the Windows/PM screen, the top can be covered by another window, and the latest information remains visible.

Since WINAUX and PMAUX run in two rather different controlling environments, you would expect the code to differ as well. Nevertheless, the two programs have many common parts. In particular, the TTY display code has been written to serve both PM and Windows. It would have been possible to create a single source for both, but as our goal is edification and explanation rather than efficiency, most of the modules are presented separately.

Since the source covers two programs and is fairly long, it will not be possible to show everything here. (For complete listings, obtain the PJ disk for this issue.) On the other hand, the offering is fairly rich. The sources and documentation for both programs are provided. In addition, we are including the Windows/PM common source for the TTY window, which we shall use frequently in future articles and which can be adapted to a variety of applications.

To demonstrate the programs, we also provide two simple templates, one in PM and the other in Windows, which have been slightly modified to drive WINAUX or PMAUX but which can be used as a basis to develop new programs in Windows or PM. Finally, we provide the include files for using WINAUX or PMAUX, which you can easily incorporate into your own programs.

What should you know in order to read this article? Some familiarity with either Windows or PM would be helpful, as we are going to concentrate here on features peculiar to WPMAUX rather than try to explain the basic organization of Windows or PM programs. Please try reading through our previous articles which have appeared in *PJ*; you may also want to look at Charles Petzold's book, *Programming Windows*. (See references on page 66.) Petzold is an excellent expositor; he leads you by the hand at every step, especially at the beginning where the learning curve is most steep.

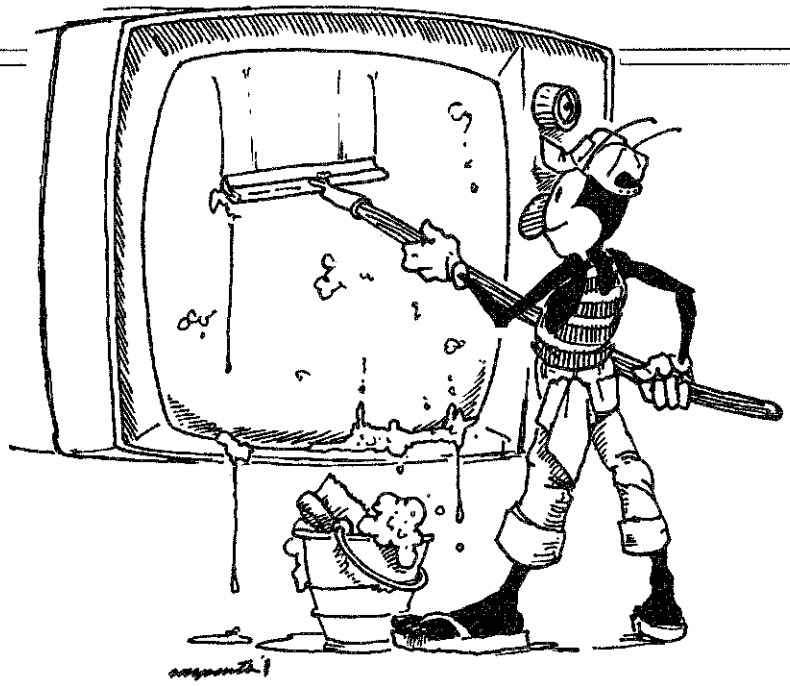
Since most of you are more likely to have the tools and experience to program Windows rather than PM, we are going to concentrate somewhat more on WINAUX. But we are not going to slight PM by any means, and if this is your main interest, you will not be disappointed.

How WINAUX and PMAUX Work

Both PM and Windows are message-passing systems. This means that most of the information for a program arrives as a result of a call by the controlling environment to one of the window procedures maintained by the program. Messages can include mouse and keyboard input, window sizing and movement data, paint requests, and a variety of other items. Messages have two information fields (in Windows, a 16-bit field wParam and a 32-bit field lParam; in PM, two unsigned 32-bit fields, usually designated mp1 and mp2) whose meanings vary with the message. In addition, each message has the handle of the target window as well as the message number itself.

Not only can a window receive messages, it can send them, too. Such

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



messages can be those predefined by Windows/PM or defined by the application. Messages can be sent from one window to another window or to the originating window, and the target window can be running in the same or another process.

For example, the easiest way to simulate the user's selecting a menu item is for the program to send itself a WM_COMMAND message containing the appropriate menu item. In a Windows or PM communication program, after reading the data from the communications port into a buffer, the buffer's pointer and length can be posted to the window via a WM_USER message. The first information parameter of the message might be the number of characters read, and the second, the buffer pointer.

Two vehicles are used to send messages: **SendMessage** (**WinSendMsg** in PM) and **PostMessage** (**WinPostMessage**). The difference between them is important. In the first, the call goes directly to the target's window procedure and does not return until completed (a value can also be returned). In the second, the message is posted to the target's message queue. If the **SendMessage** technique is used, the caller can be sure that its data segment will not move and that current automatic variables remain valid.

In contrast, posting a message implies that any data that is referenced must have meaning even after the stack frame of the caller has disappeared. Furthermore, in PM it may be necessary to synchronize the use of common data shared by two processes with semaphores or other techniques.

WINAUX expects to see a message from another program arrive in the following format:

```
Handle      WINAUX's window handle
Message     WM_USER
wParam      length of a character
            string
lParam      a long pointer to the
            character string
```

PMAUX is looking for a message in the following form:

```
Handle      PMAUX's window handle
Message     WM_USER
mp1         length of a shared text
            buffer
mp2         a segment selector for
            the buffer
```

Upon receipt, the string is copied to a local buffer and then displayed on its TTY window.

In Windows, the sending program prepares the buffer and uses the following call:

```
SendMessage(hWnd,WM_USER,(WORD)buflen,
                (LONG)(LPSTR)bufptr);
```

In PM the call is:

```
WinSend-
Msg(hWnd,WM_USER,(MPARAM)buflen,
                (MPARAM)bufsel);
```

where "bufsel" is a selector to the buffer.

How this string is prepared is up to the calling program. The easiest way is to simply use **sprintf** to format the buffer. However, if the caller is a dynamic link library with no stack segment of its own, other formatting functions may be required since the **printf** family (at least in Microsoft C) assumes ss = ds. A list of those functions that operate correctly when ss != ds is provided in the documentation for the compiler.

So, how does the caller get the window handle? When WINAUX executes, one of its last initialization actions is to write its handle to WIN.INI under [Winaux], and when it exits, it replaces the handle with 0. The calling program needs only to read the handle from WIN.INI and then test it for validity just prior to sending the message. Similarly, PMAUX leaves its handle information behind in OS2.INI under [PMaux].

In practice, the whole operation is made simpler by using the include file shown in Listing 1. For example, suppose we want to monitor the WM_CHAR messages in a Windows program. At the top of the program before **WinMain**, we include auxprt.h. Then, in the message procedure, we write:

```
case WM_CHAR:
    sprintf(auxbuf,"wParam = %x,
    lParam = %lx\n",wParam,lParam);
    auxprt(auxbuf);
    break;
```

The call to **auxprt** takes care of the details of sending the message to WINAUX. It may be useful to add a static variable in your program that is incremented after each call to **auxprt** and is displayed as part of the string. It helps to differentiate the various messages as they are displayed. Be sure to compile your program with WINAUX defined.

Although the details of formatting the message and calling **auxprt** is the same in PM, it is more complicated to pass the buffer. Since the CPU is in protected mode, the buffer must be capable of being shared among processes. There are many ways to do this. Our method is simple and can be adapted to Windows if desired; again, the details are in Listing 1. When **auxprt** is ready to send the message buffer to PMAUX, it retrieves

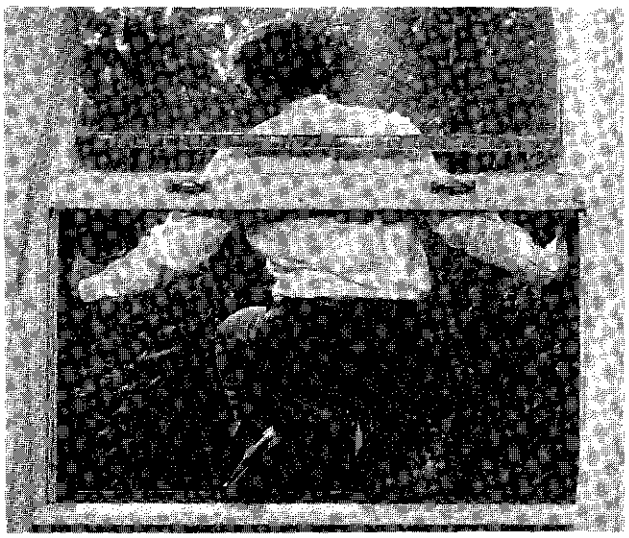
PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

the handle and tests it for validity. Next, a shared buffer is created and filled. Finally, rather than the buffer pointer being passed, the selector is sent. When the call to **WinSendMsg** returns, the shared buffer is destroyed. It seems to be a lot of overhead for each call, but it is really quite foolproof. The application has no housekeeping to do; it is all done by **auxprt** when the message is sent to PMAUX. However, be sure to compile your PM program with PMAUX defined.

Both WINAUX and PMAUX offer a carriage-return-on-line-feed option. This is a convenience for the user, since it is not necessary in every use of **sprintf** to insert a \r in addition to a \n. But WINAUX has a feature not found in PMAUX. In PM, you can run PMAUX whether or not you plan to use it immediately, size the window, position it in a convenient place, minimize it if you desire—and leave it there until you need it. It will remain in place until you reboot or shut down the machine. In contrast, you frequently enter and exit Windows while developing a new program. Hence, it is convenient to have WINAUX read a user-determined initial position and size from WIN.INI and position itself to avoid the nuisance of moving the window about on every run. WINAUX does this: The very first time it executes, it uses default values determined by Windows and writes them out to WIN.INI. The user can then change these parameters with a text editor, and WINAUX will place itself accordingly on subsequent runs. For convenience, you should add WINAUX to either the load field or the run field of WIN.INI as well.

To simplify the discussion of the programs, we have provided only the options mentioned above. But there are several others that would be useful. One is to have an optional line counter in WINAUX or PMAUX so that you can identify similarly appearing lines. A related convenience would be a clear-screen function. A very important addition would be the ability to have display pages so that the user can look back at information that has scrolled from the screen. Logging to a file could be implemented. The positioning and sizing code, already in WINAUX, could be added to PMAUX. In both, a dialog box allowing the user to choose the size and position parameters interactively would be nice for Windows and necessary for PM (since OS2.INI is *not* a text file and cannot be easily edited). If you do implement any of these (which would provide you with an excellent learning exercise), send me a copy.

Ready to jump because of your windowing software?



Frustrated with the limitations of your windowing package?

ÆWINDOS is a high-powered set of tools designed to interactively create windows at light speed with both DOS and UNIX* systems.

Written entirely in portable "C", it includes the complete source code and even make files, so changes are quick and easy.

Don't wait until you're caught in a bind, call us today. Modular, portable, and efficient, ÆWINDOS won't let you down.

aesoft

1-800-634-5494 BBS (303) 449-5714

*UNIX is a trademark of AT&T.

Number 220 on the Reader Service Card

Comments on the Source Code

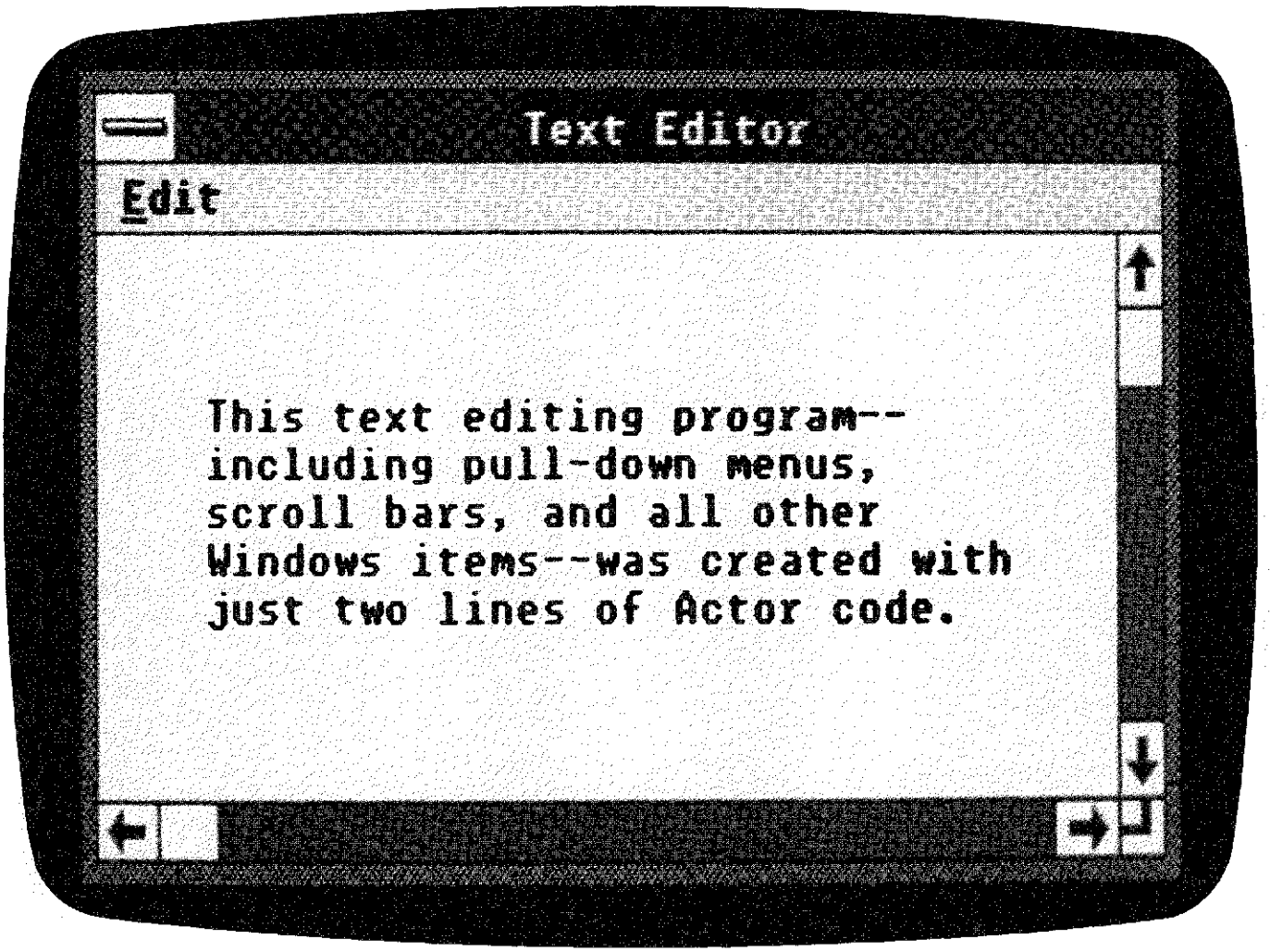
Now let us look at the main features of the source code. Because we must cover two programs, only excerpts are provided. Let us begin with the make file for WINAUX, shown in Listing 2. The corresponding file for PMAUX is very similar. It differs only in the use of protected mode libraries, the names of modules and segments, the admission of 80286 instructions, and the name of the link utility.

The WINAUX Make File

We mentioned that the TTY display code is shared between the PM and Windows versions. We maintain this code in a separate directory containing files that can be used by both programs. The location of this directory is defined in the macro **ttydir**. The directory contains a C source file, ttycls.c, and a header, ttycls.h, which has the data structures and definitions needed by any program using this source. Later, we shall cover those files in more detail.

The compile command, defined as **cp**, uses warning level 3. This is a nuisance in Windows unless you define NOCOMM in your sources (unless, of

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



Actor™ is the fastest, easiest way to develop applications for Microsoft® Windows. The reason—Actor's reusable toolkit of objects such as dialog boxes and edit windows. It more than doubles your overall productivity, making Actor an essential part of any Windows development project.

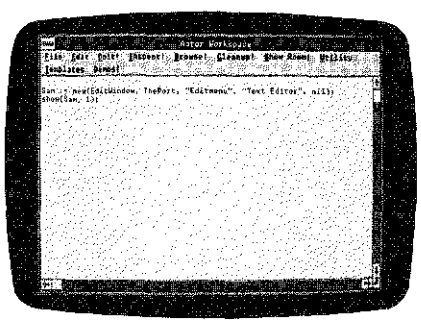
A full-featured, interactive, Windows-based programming environment, Actor provides immediate compilation, interactive testing, and source-code debugging. You can use it to produce fast standalone applications that support all Windows features, including Dynamic Data Exchange (DDE) and expanded memory.

Actor is a powerful, pure object-oriented programming language. It's all you need to develop complete Windows applications. You also have the option to dynamically link Microsoft C code to your Actor program.

Either way, it's the fastest way to produce everything from prototypes to complete development projects. No wonder so many developers are already using Actor.

Call us now for more information.

The sooner you do, the sooner you can speed up *your* Windows development work.



These two lines of Actor code are all it took to produce the Windows text editing program you see above. Just think what you can do with a few more lines.

Find out about Actor.
The Whitewater Group, (312) 491-2370

Technology Innovation Center
906 University Place, Evanston, IL 60201

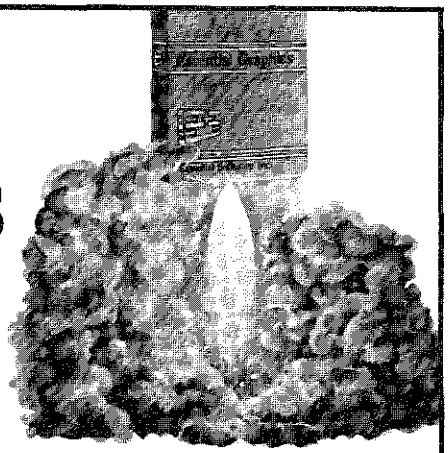
© Copyright 1989, The Whitewater Group

Number **73** on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Power Graphics

Essential Graphics Takes You To New Heights Of Graphic Programming In C. Increases Speed 40%.



When first brainstorming this ad I spent a considerable amount of time trying to determine what graphic image to use as an illustration. The Space Shuttle, Mona Lisa, Robo-Cop - there are so many available.

Then it occurred to me. When you have the fastest, smallest functions, it's really irrelevant to show a complicated graphic image. It would be as if thinking up a sexy graphic were the test of a library.

The Graphics Test

The crucial test of a professional graphics package is: are the functions powerful, reliable, fast, and do they truly eliminate grunt work?

How quickly the functions execute is the criterion most people look for in a graphics library. There is no sense paying for a package that is not up to speed.

Beware Of Speed Traps

We eliminate the bios calls and write directly to the graphics card. As a matter of fact, in a recent benchmark, we were clocked 40% faster then our nearest competition.

I'd like to repeat that "...clocked 40% faster than our NEAREST competition." Please take a moment to think about the significance of that speed increase in the project you are contemplating or working on now.

Our efficient, granular coding provides you with code sizes up to 75% smaller. Lean, fast and tight - just the way you would have done it yourself.

Power Packed Pixels In Every Package

There has always been a trade-off in this industry between ease of use and power. Our functions do not require a lot of setups, are well-documented, and most of all, thoroughly debugged. Essential Graphics' ease of use stems from our thoughtfulness and not from a lack of power. We explain what we are doing every step of the way. Our support staff consists of the humans who wrote the functions, so we are thoroughly prepared to assist you after the purchase.

Essential Graphics is a trademark of Essential Software

Caveat Emptor

Make no mistake, this is not a package for the "draw a box around the total field" crowd. This library was designed to help the professional C programmer make money and look good.

We've included a complete set of "rubber-banding" functions. One of the most welcome features is the ability to save/restore images in PC Paintbrush format or bit image. World coordinates and view ports aid in programming portability.

We include the ability to manipulate and rotate character fonts and symbols. You can place characters and symbols anywhere on the screen, and use up to eight fonts at one time.

Yours, Mine, Ours

We don't consider ourselves equity partners in your business and therefore we do not charge any royalties or run time fees. We think your efforts belong to you. If for any reason you are unsatisfied with our product you may return it within 30 days for a full refund. Full source is available. Please call today and launch yourself into the world of power graphics.

Price $299 - Source $299

Adaptors include - CGA, EGA, VGA, MCGA, ATT, ATT DEB, Hercules, Vega Deluxe, Paradise Autoswitch. **Printer Support** - IBM, Epson, Oki, TI, Alps, Panosonic, and others. **Supports** mice, light pens, plotters, color printers. **Compilers**-Microsoft, Lattice and Turbo-C

Other Essential Products Include: ScreenStar - Essential Communications and Utilities -- /*resident_C*/ - Please call for further information 201-762-6965.

To Order Call:
1-800-451-6174
N.J. 201-762-6965
Fax. 201-762-0118

Essential Software, Inc.
South Orange Plaza
76 S. Orange Ave., Suite 3
South Orange, N.J., 07079

Number 10 on the Reader Service Card

course, you need communications functions), because the bit fields for the comm data structures are defined in terms of byte instead of word quantities. This is historical. The communications driver in Windows, COMM.SYS, was written using byte-size bit fields, and if the corresponding fields in windows.h are changed to meet the ANSI standard, Windows communications programs compiled with the changed header no longer work.

Next is a library macro. Since I may need any of the various memory sizes with any of the floating-point options, I do not make the Microsoft composite libraries, which are numerous and quite large. I prefer to name each library needed and use the NODefault option, but you can change this to suit your own environment.

Notice that the medium model is specified in the compile and library macros. The medium model is the most convenient for Windows and PM. It is the easiest for creating moveable, discardable segments, which is so important for efficient memory management. The medium model has one minor disadvantage: All function calls default to FAR, and you must make specific NEAR declarations for intrasegment calls.

Four code modules in two named segments make up the bulk of the source. Note that the module **winauxnt.c**, which contains the initialization code, is in its own segment. In the definitions file, this segment is marked as PRELOAD, MOVEABLE, and DISCARDABLE. Once executed, it is never needed again, and the memory it occupies will eventually be reused. You can watch this occur if you run the HEAPWALK utility supplied with the Windows SDK.

The PMAUX Definitions File

Listing 3 shows the .DEF file from the PM version of the program. (If you have never seen a definitions file, see the *PJ* articles listed on page 66 for an explanation.) The WINAUX version is quite similar. In Windows the stub program is Winstub.exe, whereas in OS/2 it is os2stub.exe. Also, the minimum suggested stack-size parameter in PM seems to be 8KB (4KB in Windows). The poor state of documentation in OS/2 and PM often results in conflicting information. The figures given here are based on reading the source code of the applications supplied with the PM software development kit.

In both PMAUX and WINAUX, we not only have a windows procedure, we

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

also have a so-called "about" box. This is a simple *dialog box* containing static text that provides the user with basic information about the application. A dialog box is a pop-up window usually containing one or more child windows (called *controls*), with which the user can interact. A special window function called a *dialog box procedure* is required in the application. Since Windows or PM will be calling this procedure, it must be exported, just like the main window procedure.

Resources

Resources are those objects that are used to interact with the user. The resource source file contains textual descriptions of each of the objects required. Both WINAUX and PMAUX contain similar objects: strings, a menu, and an "about" dialog box. Nevertheless, there are several interesting differences (see listings 4 and 5).

Some differences are superficial. In PM, although it is not documented, it is becoming customary to replace the BEGIN and END statements with "{" and "}", respectively. (You can do this in Windows, too.) Another small difference is the way menus and dialog boxes are identified for later use by the application. In Windows, menus have been named by character strings and loaded at class registration time. In PM, the menu is simply identified by a constant value, which is passed to the window create function, and, although I have not yet tried it, apparently you can nest menus in PM.

Another difference between PM and Windows is the way dialog box templates are described. In PM, a rectangle is labeled by points specifying the lower left and upper right corners; in Windows, the points are the upper left and lower right corners. Hence, in the DIALOG statement, the pair 10,60 refers to where the lower left corner of the box will appear in the window. In Windows, the first pair locates the upper left corner.

A similar difference arises in positioning the various static text windows in the "about" box template. For example, the OK button in WINAUX is located at 61 dialog units to the right and 77 dialog units down from the upper left corner of the box. In PM, the same button is 64 units from the left and 4 units up from the bottom left corner of the box.

You may notice that some items in the dialog box and menu statements use identifiers such as -1 or 0. These are throwaway values used to fill a required field in the template. However, any control in a dialog box or menu that is going to be referenced in the application must have an identifier unique to the template. For example, DID_OK, a predefined value, is used to refer to the OK button in PM.

Although a mouse is very useful in Windows and PM, all commands can also be executed with the keyboard. For the user's convenience, an accelerator key is often defined for each menu item. You can specify an accelerator in PM by prefixing a character with the tilde (~); in Windows, you use the ampersand (&). You can see examples in the MENUITEM statements. When the menu is visible, the corresponding characters will be underlined and can be accessed quickly with the Alt key; for example, pressing Alt-O-A will display the "about" box.

The Window Procedure and Support Functions

Listing 6 shows all of the module WINAUX.C. PMAUX.C is contained in Listing 7. Here we see the entry point and the message function for each program. Notice that fewer messages are acted upon in the PM version; we will contrast the two by analyzing some of the messages that are processed. It may

SuperFAST!

C-Index™

Microsoft C Turbo C

Unix VAX Mac Windows OS/2

Database Toolkit

Nine simple function calls give you the power of fast B+Tree indexing and automatic variable length records.

C-Index makes data management programming easy with a friendly Application Program Interface, excellent documentation, extensive examples and a tutorial database application. C-Index is ready-to-use with Microsoft C or Turbo C, right out of the box. No need to compile or change the source code.

C-Index is the fastest database toolkit available, up to ten times faster than C-Tree and CBTree! C-Index is also the most flexible package you can buy with 9 high level functions and 37 building-block functions that allow you to tailor C-Index to every program.

C-Index is the library of choice for thousands of software developers. C-Index has been successfully used in mass market commercial applications, such as **Prime Time,** the highly acclaimed personal time manager. Wayne Ratliff, author of dBase II and dBase III, thought C-Index was so "terrific" that he uses it as the foundation for his new **Emerald Bay** database system.

Discover for yourself the power of C-Index. Call to order your own copy today.

- No Application Royalties
- Complete Source Code
- Single and Multi-user Access
- 170 page manual
- Proven Reliability
- Fast B+Tree Indexing
- Variable Length Records
- Random and Sequential Access
- Data and Indexes in same file
- Multiple Record Formats per file

"I heartily recommend this package."

Dr. Dobb's Examining Room, June '88

C-Index™ for Turbo C® **$99.00**

Complete source and libraries for Turbo C 1.5 and 2.0

C-Index™ for Microsoft C® **$195.00**

Complete source and libraries for Microsoft C versions 5.0 and above.

C-Index/Plus for Unix/Xenix and All Other Systems **$395.00**

Includes both above libraries plus all source code and test routines required to port to any system: Unix, VAX, Mac, Windows, DOS and OS/2.

Trio Systems **213/394-0796**

2210 Wilshire Bl. Suite 289 Santa Monica, CA 90403

Number **262** on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

help you to see exactly how each message is handled. Extracts from the support modules WPMAUXFN.C and PMAUXFN.C are presented in listings 8 and 9, respectively.

Recall that in both WINAUX and PMAUX, the next line to be displayed is placed at the bottom of the client area. Now, in PM, the lower left corner of the client area is the origin, so writing to (0,0) will produce the same result even if the window is resized. But, in Windows, resizing may cause the coordinates of the lower left corner to change. Hence, in Windows we must listen for WM_SIZE; when it is received, the display point must be adjusted and the client area redrawn. **AdjustHeight** accomplishes this goal by changing the current point in the MWnd data structure. This structure will be studied in more detail when we look at the TTY code.

Both programs process the WM_COMMAND message to handle the menu by calling **WndCommand**. The PM and Windows versions of this function are very similar. The response to the IDM_CRONLF item is to toggle the corresponding parameter in the MWnd data structure and to check or uncheck the menu. When the IDM_ABOUT item is received, the usual code for creating a modal dialog box is executed. The call does not return until the dialog box is closed in the dialog box procedure; hence, the program can accomplish no other task until the OK button is "pressed."

Both programs use WM_USER to start the process of displaying the string. However, the **DispatchString** functions differ somewhat in their parameters and how they process the information received. In WINAUX, the string is copied to a local buffer and simply sent on to **TTYDisplay**. However, the PM version of this function must first convert the selector passed in mp2 to a long pointer with MAKEP and make a specific call to access the associated buffer. Only then can it read the data to a local buffer and send it on to **TTYDisplay**.

Closing an application is more elaborate in Windows than in PM simply because it can be accomplished in two ways: by terminating the program or by ending the Windows session. With the former method, a WM_CLOSE message is sent, and the application can then take a variety of steps, even as far as preventing immediate shutdown. The latter method is more complicated. First, a WM_QUERYENDSESSION message arrives. If all current applications return nonzero, then the session can end. If one returns zero, then the session cannot end until nonzero is returned. During that time, other applications that have agreed to end get WM_ENDSESSION messages with wParam = 0. Finally, when wParam arrives with a nonzero value, the session is really terminating.

Since a WM_CLOSE message is not received when the session is ending, an application must be prepared to close using either method of terminating. So, in WINAUX, we carry out in two places the step of calling **SetWinIni** to restore the window handle to NULL. By contrast, there are no analogs to the WM_ENDSESSION class of messages in PM; it is enough in PMAUX to reset the handle in OS2.INI and post a quit message. Note the great similarity of the functions **SetOS2Ini** and **SetWinIni** in listings 8 and 9.

WM_PAINT is one of the most important messages in Windows or PM. It is a signal that the client area has been modified in some way—perhaps by another window, dialog box, or menu—and that an update is needed. Hence, WM_PAINT arrives complete with a handle to a display context (in PM, presentation space), and an update rectangle. It is up to the application to choose how the message is processed, except that each response must start with **(Win)BeginPaint** and finish with **(Win)EndPaint**.

In our programs, we take different actions depending on whether or not the window is iconic. If it is, we simply draw a border rectangle and write an abbreviated form of the program's name in the icon area; otherwise, we call on the code for the TTY window to redraw the text lying under the update rectangle. Since the program keeps a buffer of the most recently received text, we simply translate the update rectangle coordinates into corresponding character positions in the buffer.

Actually, we do not worry about horizontal boundaries of the rectangle but simply blast out the entire line if it has been affected anywhere. We are, however, interested in limiting the number of lines actually redrawn, and we use the top of the update rectangle to avoid drawing those lines that lie above and that do not need repair. This is especially useful when the window is scrolled upward. Normally, such a scroll is carried out by a bit-block transfer (bitblt), which moves the window upward intact

The
Heap Expander™
version 2.5

Now your programs can have virtually unlimited heap space using expanded memory, extended memory, disk space, or any combination of the three. And it's all transparent. The Heap Expander's initialization code checks the system's resources and uses whatever is available.

only $79.95*

- Uses LIM-standard expanded memory if present.
- Uses AT-style extended memory if present.
- Swaps data to disk as needed.

Libraries and Source Code for:
- Turbo C
- Microsoft C 4.0 and 5.0
- Turbo Pascal 3.0 and 4.0

Requires an IBM PC, XT, AT, or close compatible with MS-DOS or PC-DOS version 2.0 or above

MC/VISA/COD call
1-800-248-1045 x 100 (US)
1-800-952-5560 x 100 (Idaho)

*CA residents add 6 1/2% sales tax.
Foreign customers, please add $4.00 for shipping and handling.

The Tool Makers
75 Mt. Hermon Road
Scotts Valley, CA 95066
(408) 458-0690

Number 166 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

and leaves only the just-uncovered bottom line to be updated. (The details of how the icon is drawn can be seen in our previous articles. The text-redrawing function can be found in the code for the TTY window.)

We conclude this section by showing the simple dialog box procedure needed to handle the "about" box. In Windows, a dialog box function is a kind of preprocessor to the actual window procedure, which itself is external to the application. By contrast, in PM the dialog box handler is written just like any other window procedure. Thus, if a message is acted upon, then FALSE is (usually) returned, whereas all unprocessed messages go to a default dialog box procedure. Contrast this with the Windows version, which returns FALSE if the message is *not* processed and TRUE if it is acted upon. You can compare the two in listings 8 and 9.

WINAUX Initialization

The initialization module for WINAUX is given in part in Listing 10. It is mostly boilerplate—the window class is registered, the window is created and shown, and, finally, its handle is written to WIN.INI. Only one instance of the program is permitted; if the instance handle is not NULL, the program exits. To be a bit more user-friendly, you might want to place a message box announcing something like "WINAUX already running" at the appropriate place.

There are two features in this module that need further comment. The first is the call to **getcrect**, of which we have given an excerpt. Recall that WINAUX can read WIN.INI in order to place its window. Here we try to read an integer under the AppName "Winaux" with keystring "x". The default value is set to -1 so that CW_USEDEFAULT is written to WIN.INI if a negative value is returned. This step is repeated for the *y* position as well as the horizontal and vertical extents of the window. The user can then easily edit the entries to adjust the starting position and size of the window.

The second feature is a bit more subtle and rather interesting. When a window is registered, space can be allocated for extra data that is common to the class (class extra) or unique to each window of the class (window extra). In WINAUX, we allocate two extra bytes of the latter type to hold a short pointer to a TTYWND data structure. This structure, which can be seen in Listing 11, contains the data needed to manage a teletype-style window.

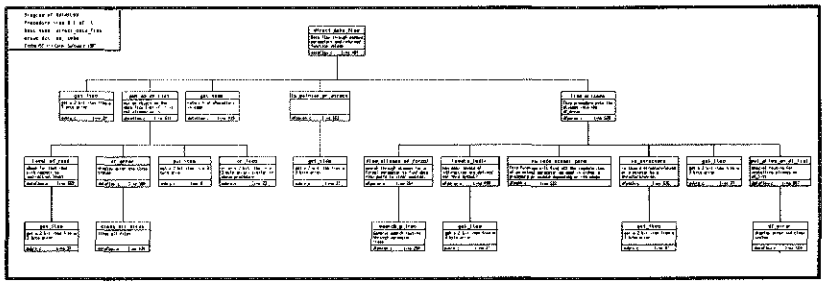
When the window is created, we place a pointer to the TTYWND data structure associated with that window in the last parameter passed to **CreateWindow**. Then, when the WM_CREATE message is received, the data structure is retrieved and initialized. For our purposes, the most important initialization is the creation of a buffer to hold the text along with all the pointers and data needed to display and redisplay the text and to perform carriage returns, line feeds, tabs, and so on. By the time the **CreateWindow** function has returned, WM_CREATE has been processed. In particular, the buffer pointer field has been initialized, and if it is not NULL, the program can continue; otherwise, it aborts.

In the TTY code itself, references to data are made only via the pointer to the TTYWND data structure, which is always passed in by the calling code. Hence, both the initialization and the main window support code are quite independent of any particular window.

Of course, only one window is needed in this project; however, the techniques used here are very useful for managing many child windows of the same class. When a message is received, a call to **GetWindowWord** with the appropriate

C/ANALYST...

...THE HIGH POWERED DOCUMENTATION TOOL



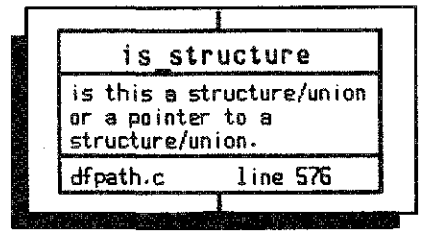
You have been handed a few inches of source code and have been asked to modify it. C/ANALYST can quickly bring you up to speed. It analyzes your C source and forms a database representing your entire program. From this database, C/ANALYST exhibits your program from different perspectives.

Currently C/ANALYST is a set of four programs: ANALYZE, XREF, PROCTREE & DATAFLOW.

ANALYZE forms the database by reading in multiple C source files. It has a built-in macro preprocessor which will perform macro replacement, retrieve headers, and conditionally include sections of code. It will then tokenize the input and parse it. Information stored in the database includes which functions are called by a particular function, what parameters are passed to the function, and what variables are set or referenced in a function. Virtual memory is used, so the database can be much larger than available memory.

XREF generates a function index followed by a global cross reference listing. The function index alphabetically lists each function, gives its location and its definition taken from comments in the source code. The global cross reference not only shows where each object is used but how it is used as well. Read and Write codes along with indirection levels are included with each reference. Objects are referenced to the function which uses them. Structure members are listed individually.

PROCTREE generates procedure tree diagrams which show the overall control flow of the program. They are printed sideways on a dot matrix printer using overlapping dots. This makes them much easier to read. Although the diagrams can be almost unlimited in size, a heuristic pruning method chops trees into multiple diagrams of reasonable size.



DATAFLOW searches through the database to find the flow of information between each module pair in the program. It shows flow through global variables when one module writes to a variable and another module reads that variable. Parameter passage to functions can be shown along with symbolic function returns. A sophisticated technique searches for indirect data passage through formal parameters which are pointers.

FOR MORE INFORMATION CALL US TODAY

(503) 581-5622

$150.00 post paid
+ 10.00 for overseas

30 day money back guarantee
requires MS-DOS 2.0 or above.

CATER SOFT WARE

2182 Westfarthing Way NW
Salem, OR 97304

Number 263 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

FINALLY!

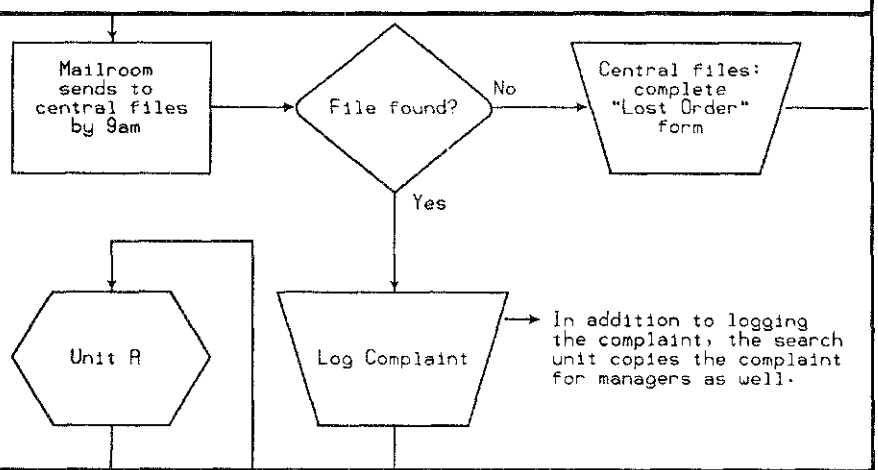
EASYFLOW

An on-screen flowchart processor that knows about flowcharts - not just another "screen draw" program that makes you do most of the work. EASYFLOW is a powerful full-screen graphics program dedicated to flowcharts and organization charts. With it you can quickly compose charts. More important, you can easily modify charts so they are always up to date.

- **Automatic:** Fully automatic text centering within shapes, both horizontally and vertically. Fully automatic line routing & re-routing.
- **Fast:** Written in assembly language for speed.
- **Large:** Charts up to 417 columns wide by 225 lines high. Chart too large for your printer? *EASYFLOW* automatically breaks the chart up & prints it in page size pieces.
- **Standard:** All standard flowcharting shapes included. Custom shapes can be ordered.
- **User friendly:** Don't take our word for it. PC Magazine says *"EASYFLOW lives up to its name. It's hard to imagine any easier and more flexible way to produce basic and even complex flowcharts".* [March 10th 1987 issue, page 278.]
- **Mouse:** Optional but fully supported.
- **It prints:** On most popular matrix printers including IBM, Epson, Toshiba, HP LaserJet, LaserJet-Plus and many others.
- **It plots:** On HP7440, 7475, 7550, 7570, 7585B and compatible plotters.
- **It works:** We are contractually prevented from mentioning the name of the "big eight" accounting firm that purchased a world-wide site license, but we can tell you that they spent months evaluating all available flowcharting packages before choosing *EASYFLOW*.
- **Rush delivery:** Order by noon today (eastern time) and you'll have it tomorrow. Rush delivery charge is $10.50 (instead of $2.00) and is available only to continental USA and Canada.
- **Documented:** 100 page manual plus over 150 screens of context sensitive help.

EASYFLOW works on IBM PC's, IBM PS/2 and compatibles. Requires 384 K memory, DOS 2.0 or higher and an IBM CGA/EGA/VGA or Hercules monochrome compatible adapter card.

Order direct for only **$149.95** + $2.00 S&H (USA/Canada), $10.00 (foreign). Payment by M.O., cheque, VISA, Mastercard or Company PO.


Mailroom sends to central files by 9am
File found?
No
Central files: complete "Lost Order" form
Yes
Unit A
Log Complaint
In addition to logging the complaint, the search unit copies the complaint for managers as well.

The chart fragment above was produced on an HP LaserJet-Plus and is actual size and unretouched. Publication quality charts like this can be produced using only minutes of preparation time and seconds of print time.

HavenTree Software Limited
PO Box 1093-V
Thousand Island Park, NY 13692

Order Desk: 1-800-267-0668
Information: (613) 544-6035 ext 31
Telefax(G3): (613) 544-9632

Number **154** on the Reader Service Card

index will fetch the pointer to the data structure that applies to that particular child. The last function procedure in Listing 10 shows how the text buffer is created and the window extra data is initialized to a pointer to the windows's TTYWND structure. First, to determine the size of the buffer, we retrieve the maximum width and height of the window. Next, the system font size is measured. Then, the TTYWND structure is retrieved and used to initialize the extra data. Finally, a call is made to the TTY code module to create the text buffer. The last four parameters in **InitTTYWindow** simply specify word wrap, line feed on a carriage return, carriage return on a line feed, and a character mask (usually 0x7f or 0xff).

Initialization of PMAUX

Although it's not shown here, PMAUX's initialization follows roughly the same path as that of WINAUX. In particular, the **WndCreate** function, which is executed when WM_CREATE is received, calls the same **InitTTYWindow** function. Additionally, a window must be registered and created as in Windows, although the details differ. You can see how the basic steps are carried out in "An OS/2 Presentation Manager Application Template" (*PJ* 6.5).

However, there are differences in initializing WINAUX and PMAUX. In PMAUX, a message queue must be created and an anchor block handle retrieved. Also, we do not try to place the window initially, prevent more than one instance, nor use the extra data field to pass the TTYWND data structure to **WndCreate**. Such omissions are partly due to our ignorance at this time of the exact Windows parallels and the extent to which the methods we use there can be extended to PM. Some of our problems are directly related to the really disgusting quality of the documentation in PM. And although there is on-line help, it is also full of errors. Combine this with the continuing instability of the development version of PM and the ways it differs from the just-released IBM version, and perhaps you can understand our reluctance at this time to attempt some of the things we do regularly in Windows. For example, consider the following situation:

In WINAUX, the text buffer is created and maintained in the heap using the various local memory-management functions. In PM, at least at this time, the techniques for using the local heap functions are quite obscure. In

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

C CODE FOR THE PC

source code, of course

	MS-DOS File Compatibility Package (create, read, & write MS-DOS file systems on non-MS-DOS computers)	$500
	Bluestreak Plus Communications (two ports, programmer's interface, terminal emulation)	$400
	CQL Query System (SQL retrievals plus windows)	$325
	GraphiC 4.1 (high-resolution, DISSPLA-style scientific plots in color & hardcopy)	$325
	PC Curses (Aspen, Software, System V compatible, extensive documentation)	$250
	Greenleaf Business Mathlib (exact decimal math, formatting, depreciation, interest, cash flow, statistics; specify compiler)	$250
	Greenleaf Data Windows (windows, menus, data entry, interactive form design; specify compiler)	$220
NEW!	Assembler Kit (by John Zarrella; includes listing generator & loader; requires signed license agreement)	$175
	Greenleaf Communications Library (interrupt mode, modem control, XON-XOFF; specify compiler)	$175
	TurboTEX (TRIP certified; HP, PS, dot drivers; CM fonts; LaTEX)	$170
	Sherlock (C debugging aid)	$170
NEW!	AT BIOS Kit (roll your own BIOS with this complete set of basic input/output functions for ATs)	$160
	Greenleaf Functions (296 useful C functions, all DOS services; specify compiler)	$160
	WKS Library Version 2.0 (C program interface to Lotus 1-2-3, dBase, Supercalc 4, Quatro, & Clipper)	$155
	OS/88 (U**x-like operating system, many tools, cross-development from MS-DOS)	$150
	ME Version 2.1 (programmer's editor with C-like macro language by Magma Software; Version 1.31 still $75)	$140
	Vmem/C (virtual memory manager; least-recently used pager; dynamic expansion of swap file)	$140
	Turbo G Graphics Library (all popular adapters, hidden line removal)	$135
	TurboGeometry (library of routines for computational geometry)	$125
	CBTree (B+tree ISAM driver, multiple variable-length keys)	$115
	TE Editor Developer's Kit (full screen editor, undo command, multiple windows)	$105
	Minix Operating System (U**x-like operating system, includes manual)	$105
NEW!	HyperText Viewer (simple hypertext system; multi-file documents; includes Tiny Curses)	$100
	PC/IP (CMU/MIT TCP/IP implementation for PCs)	$100
	B-Tree Library & ISAM Driver (file system utilities by Softfocus)	$100
	Tele Operating System (TeleKernel, TeleWindows, TeleFile, & TeleBTree by Ken Berry)	$100
	The Profiler (program execution profile tool)	$100
	QC88 C compiler (ASM output, small model, no longs, floats or bit fields, 80+ function library)	$90
Updated!	Install 2.0 (automatic installation program; user-selected partial installation; CRC checking; Version 1.0 still $25)	$80
	Wendin Operating System Construction Kit or PCNX, PCVMS O/S Shells	$80
	C Windows Toolkit (pop-up, pull-down, spreadsheet, CGA/EGA/Hercules)	$80
	JATE Async Terminal Emulator (includes file transfer and menu subsystem)	$80
	Polyglot Lisp-to-C Translator (includes Lisp interpreter, Prolog, and simple calculus prover)	$80
	MultiDOS Plus (DOS-based multitasking, intertask messaging, semaphores)	$80
NEW!	Kinetic Image Synthesizer (3-D animation system ... Saturday morning on your PC!)	$75
	XT BIOS Kit (roll your own BIOS with this complete set of basic input/output functions for XTs)	$75
	Professional C Windows (lean & mean window and keyboard handler)	$70
NEW!	Heap Expander (use LIM-standard expanded memory as an extension of the heap)	$65
	lp (flexible printer driver; most popular printers supported)	$65
	Quincy (interactive C interpreter)	$60
	PTree (parse tree management)	$60
NEW!	SuperGrep (exceptionally fast, revolutionary text searching algorithm; also searches sub-directories)	$50
	OBJASM (convert .obj files to .asm files; output is MASM compatible)	$50
	Icon-Tools (full-featured icon display and editing system)	$60
NEW!	Polyglot TSR Package (includes reminder, bookmark, virus catcher, cache manager, & speech generator)	$50
	HELP! (pop-up help system builder)	$50
	Multi-User BBS (chat, mail, menus, sysop displays; uses Galacticomm modem card)	$50
	Make (macros, all languages, built-in rules)	$50
	Coder's Prolog (inference engine for use with C programs)	$45
	Virtual Memory System (least recently used swapping)	$40
	C-Notes (pop-up help for C programmers ... add your own notes)	$40
	Heap I/O (treat all or part of a disk file as heap storage)	$40
	Biggerstaff's System Tools (multi-tasking window manager kit)	$40
Updated!	OOPS (collection of handy C++ classes by Keith Gorlen of NIH; Version 2.2)	$35
Updated!	Bison & PREP (YACC workalike parser generator & attribute grammar preprocessor; now includes documentation)	$25
	PC-XINU (Comer's XINU operating system for PC)	$35
	CLIPS (rule-based expert system generator, Version 4.2)	$35
	Tiny Curses (Berkeley curses package)	$35
NEW!	Polyglot RAM Disk (change disk size on the fly; includes utilities)	$30
	SP (spelling checker with dictionary and maintenance tools)	$30
	Clisp (Lisp interpreter with extensive internals documentation)	$30
	Translate Rules to C (YACC-like function generator for rule-based systems)	$30
	6-Pack of Editors (six public domain editors for use, study & hacking)	$30
	Crunch Pack (14 file compression & expansion programs)	$30
	Pascal P-Code Compiler & Interpreter or Pascal-to-C Translator (Wirth standard Pascal)	$25
	ICON (string and list processing language, Version 7.5)	$25
Updated!	FLEX (fast lexical analyzer generator; new, improved LEX; Version 1.1)	$25
	LEX (lexical analyzer generator; an oldie but a goodie)	$25
	AutoTrace (program tracer and memory trasher catcher)	$25
	Data Handling Utilities in C (data entry, validation & display; specify Turbo C or Microsoft)	$25
	Arrays for C (macro package to ease handling of arrays)	$25
	ANSI Forms (forms manager based on ANSI codes)	$20
	C Compiler Torture Test or Benchmark Package (checks a C compiler against K & R or measures speed)	$20
	A68 (68000 cross-assembler)	$20
	List-Pac (C functions for lists, stacks, and queues)	$20
	XLT Macro Processor (general purpose text translator)	$20
	C/reativity (Eliza-based notetaker)	$15
	Data	
	DNA Sequences (GenBank 57.0 including fast similarity search program)	$150
	Protein Sequences (5,415 sequences, 1,302,966 residuals, with similarity search program)	$70
NEW!	SIC Codes (each SIC code with the name of the industry to which it applies)	$70
	Dictionary Words (234,932 words in alphabetical order, no definitions)	$60
	U. S. Cities (names & longitude/latitude of 32,000 U.S. cities and 6,000 state boundary points)	$35
	The World Digitized (100,000 longitude/latitude of world country boundaries)	$30
	KST Fonts (13,200 characters in 139 mixed fonts: specify TEX or bitmap format)	$30
NEW!	King James Bible (Old and New Testaments)	$25
	USNO Floppy Almanac (high-precision moon, sun, planet & star positions)	$20
	NBS Hershey Fonts (1,377 stroke characters in 14 fonts)	$15
	U. S. Map (15,701 points of state boundaries)	$15

The Austin Code Works
11100 Leafwood Lane
Austin, Texas 78750-3409 USA

acw!info@uunet.uu.net

Voice: (512) 258-0785
BBS: (512) 258-8831
FAX: (512) 258-1342

Free surface shipping for cash in advance **For delivery in Texas add 7%** **MasterCard/VISA**

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

desperation, I used "grep" on every PM example supplied on the SDK for any function of this type. I found exactly one program that I could use as a guide, and the PM code following this article duplicates what I discovered. At this time, I can offer no explanation of the details except to say that it works in this release but may not in the next.

Finally, let me mention another, future difficulty. In PMAUX, the display of the received text depends rather heavily on the default (system) font in the SDK having a fixed pitch. However, the retail IBM release of PM actually has a proportional system font. Such a font will appear in the next version of the SDK, meaning that the current code for PMAUX will have to be revised. We will publish the changes for you, but as you can see, writing even the simplest applications in PM is like trying to program on a waterbed.

The TTY Window

Finally let us talk about how text is shown in the window. Both PMAUX and WINAUX share the same module. This module has three entry points: **InitTTYWindow** to create the text buffer and initialize the TTYWND data structure, **TTYDisplay** to place the text on the screen, and **TTYPaint** to handle the WM_PAINT message. Four local functions respond to the receipt of a carriage return, line feed, backspace, or tab. Once again, we apologize for not being able to show the entire module. However, Listing 11 does contain all of ttycls.h, and Listing 12 at least contains the three entry points.

We have to work harder to show text in Windows or PM than in character-based displays, in which it is enough to set the mode and place the correct value in one of the predetermined video memory slots or to send the adapter a control character and let the hardware do the rest. In a windowing environment, however, the display is always in graphics mode, and the text can appear anywhere, in any available font, in any available color, and in opaque or transparent mode. Everything must be managed in software. Not only must procedures be written to place the text into the window and respond to a reasonable subset of the control characters, but a current text buffer must be maintained to redraw the window at any time, and, if implemented, paste text to the clipboard.

Now, if you look at the TTYWND structure (Listing 11) you will see explanations of all the fields needed to maintain our simple write-to-bottom-line and scroll-upward window. The differences in the structure between PM and Windows deserve comment. The current position of the insert point is maintained in a point structure. Unfortunately, in Windows, POINT is a pair of shorts, whereas in PM, POINTL, is a pair of *longs*. In addition, the techniques for local heap management differ between the two environments. In Windows, **LocalAlloc** returns a handle to the allocated buffer, and a pointer is then obtained by passing the handle to **LocalLock**. In PM, it seems to be enough to get a handle to the heap at initialization time and keep this handle as a global variable. **WinAllocMem** can then be called to allocate the memory and return the pointer whenever a buffer is needed.

Finally, turn to Listing 12 to see the details of how code can be shared between PM and Windows. **InitTTYWindow** contains conditional compilations for handling the different ways in which local memory management is conducted. Also of interest is that when the current position is initialized, it is set to (0,1) in PM. The starting abscissa is the same in Windows, but the ordinate is adjusted for the height of the window. The difference reflects the way PM and Windows locate the origin.

The function of **TTYDisplay** is to read a string of bytes, save printable characters into the display buffer and show them, and take appropriate action on the five standard control characters. The important differences between PM and Windows appear mainly in drawing the characters on the screen. In Windows, the default display context, which is retrieved with a call to **GetDC**, is already set for writing text with a black pen in opaque mode. In PM, the presentation space (PS) has to be set to overpaint prior to calling **GpiCharStringAt**, and it should be restored before the PS is released.

The function of **TTYPaint** is to loop through the lines in the buffer and display them on the screen. To limit the number of lines actually drawn, the top of the update rectangle is made available, and this is converted to a row number by dividing the height of the rectangle by the height of the font. Any line falling in this range is then displayed. It would be possible to use the sides and bottom of the rectangle to further limit the actual number of characters redrawn. Of course, there is a tradeoff. Windows and PM automatically clip the display, and those bits lying outside the update rectangle are not actually drawn. It is not always clear how much time is saved if the application also performs some similar computations.

Summary

In this article, we have shown you one way to make a Windows or PM program send information to another. Along the way, we have developed a window class, the TTY Window, which can be used to display information in a variety of simple applications. Finally, we have explored the relationship of Windows and PM and have shown you a way in which code for an application running in both environments can be maintained.

As a senior engineer at Olivetti Advanced Technology Center in Cupertino, California, William S. Hall specializes in Windows system support software.

Code follows. ♦

REFERENCES

"A Digital Clock for Microsoft Windows" (with M. J. Babcock), *Programmer's Journal* 6.4 (1988), 40-59.

"An OS/2 Presentation Manager Application Template," *Programmer's Journal* 6.5 (1988), 44-56.

Charles Petzold, *Programming Windows*, Microsoft Press (1988).

JAKE™: A NATURAL LANGUAGE C LIBRARY
Turns English Sentences into Function Calls & Data Structures
OUR CUSTOMERS USE JAKE FOR

Script Languages	**COMMANDS**
Process Control	
Expert System Rules	
Command Languages	
Hypertext Systems	**QUERIES**
Help Systems	
Ad Hoc Database Queries	
Library Card Catalogue Retrieval	
Electronic Funds Transfer	**ANALYSIS**
Stock Market Tape Monitoring	

JAKE $495. Interactive Demo $10
CALL (408) 438-6922 VISA, MC

EKS English Knowledge Systems, Inc.
5525 Scotts Valley Dr. Suite 22
Scotts Valley, CA 95066

Usable with Microsoft C and Turbo C.
Context-sensitive semantic processing.
Easy interface to any application. Mem: 64K.

Number **243** on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

TurboC, Lattice C, and QuickC

"c_wndw" Windows, Menus & File Access

Now there is an affordable, complete set of functions to produce end-user DOS applications in the C language. "c_wndw" is a very complete toolkit, providing the following facilities

- o Color and cursor control
- o Pull-down and pop-up menus, and dialog lines
- o Field formatting and string handling
- o Entry editing of numbers and dates
- o Stacked, overlapping windows
- o File access error trapping and handling
- o Access to dBASE III plus '.DBF' files
- o Hashed random file access
- o Function key mapping and control
- o Undo and Help capability

For a license fee of $35, we provide a full set of object libraries for a C compiler, the full manual, and 19 sample programs. (S&H $2, GA res. add $1.85 tax). Specify your C compiler. Source code is available for $95.

To find out more about the "c_wndw" toolkit, write or call for information to Marietta Systems - (404) 565-1560
P.O. Box 71506, Marietta, GA 30007 Visa-M/C

Trademarks: "c_wndw", dBASE, QuickC, Turbo C, Lattice C

Number **223** on the Reader Service Card

CC-Rider

POP UP YOUR SYMBOLS

Get *Fansi* by adding ***hypertext*** capabilities to your C editing environment. Gain access to all your macro, function, structure and typedef definitions instantly!

See CC-RIDER
SEE WHAT YOU CAN DO!

INCLUDES:

CCSYM Standard C "symbol compiler" which builds the symbol database used by CC-RIDER. Supports Microsoft C v5.1 extensions.

CCRIDER Pop-up program to recall your symbol definitions in a special window from any program.

USE YOUR OWN EDITOR

Western Wares
Box C Norwood, CO 81423
(303) 327-4898 ***Demo $5***

C.O.D., VISA, MASTER CHARGE and AMEX
ONLY $89

Number **100** on the Reader Service Card

C™ Programmer's Toolbox

PC/MS DOS 2.1 & later

CFLow™	Determine your C program's function hiearchy; external R/T functions; etc.
CPrint™	Beautify C source code to your selected format
PMon™	Monitor your program/OS execution; Determine where time is being spent...
	+ 12 more tools...

Volume I

Are you creating better, faster, higher quality and easier to read programs in a fraction of the time?

Join our many beginning and expert users.

Volume II

9 more tools that you requested including:

CLint™	Perform additional syntax checks of your C source code-find silly mistakes...
CritPath™	Determine your C program's critical path from PMon and CFlow report inputs...
CXRef™	Trace your symbol/variable usage; identify potential usage problems...

The Toolbox works with all compilers and libraries. Specify any number of files through wildcards and/or indirect files.

Buy both Volumes I & II for $140, saving $20, or only one volume for $79.95. CA residents please add applicable tax. Overseas, please add $10 shipping & handling. With our 30 day money back guarantee, you can't lose.

Call or write us today

MasterCard® VISA®

C™ Programmer's McTool Series

Apple® Macintosh™

Common Tool Features

- o Powerful, built-in editor
- o Automatic/manual search & replacement
- o Selective C source code highlighting
- o Works with all C compilers
- o ANSI support

#1 **McCPrint™**	$59.95
#2 **McCLint™**	$99.95

MMC AD Systems™

Box 360845
Milpitas, California 95035
(408) 263-0781
"The C Tool Specialists"

Number **94** on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

All code in this issue is available on 360K, MS-DOS disk.
ORDER BY PHONE — VISA/MC orders call toll free 800/234-0386.
Mail order information on pages 8 and 88.

Code from "A Simple Debugging Utility" by William S. Hall

1

LISTING 1

```
 * Include file for Winaux/PMaux.
 * See WPMAUX.DOC for usage
 *
 * Written by William S. Hall
 */

char auxbuf[80];
void auxprt(char *str)
{
    HWND hWnd;
    int len = strlen(str);

/* use this section for windows */
#if defined(WINAUX)
  /* get the handle for winaux from win.ini */
    hWnd = (HWND)GetProfileInt((LPSTR)"Winaux", (LPSTR)"hWnd", 0);
  /* send over the buffer */
    if (IsWindow(hWnd))
        SendMessage(hWnd, WM_USER, (WORD)len, (LONG)(LPSTR)str);
#endif

/* use this section for Presentation Manager */
#if defined(PMAUX)
    char hbuf[40];
    SEL sel;
    PCH pchBuf;
    int i;

  /* get the string representation of the handle for pmaux from OS2.INI */
    WinQueryProfileString(hAB, "PMaux", "hWnd", "", hbuf, 40);
  /* convert to a handle */
    hWnd = (HWND)atol(hbuf);
    if (WinIsWindow(hAB, hWnd)) {
      /* create a shared buffer which can be read by another process */
        if (DosAllocSeg(len, &sel, SEG_GETTABLE) == 0) {
          /* make a long pointer to the buffer */
         pchBuf = MAKEP(sel,0);
          /* load it up */
            for (i = 0; i < len; i++)
                *(pchBuf+i) = *(str+i);
          /* send it over */
            WinSendMsg(hWnd, WM_USER, (MPARAM)len, (MPARAM)sel);
          /* free the buffer */
            DosFreeSeg(sel);
        }
        else  /* error, ring the bell */
            WinAlarm(HWND_DESKTOP, WA_WARNING);
    }
#endif
}
```

LISTING 2

```
# Make file for WINAUX, a simple debugging window for Windows

# location of scrolling window code
ttydir=..\..\pwcommon\ttycls

# header macro
headers=winaux.h $(ttydir)\ttycls.h

# compile macro
cp=cl -c -AM -Gsw -Os -Zpe -W3 -DWINDOWS -I$(ttydir)

# libraries
LIBS=mlibw mlibcr libh /NOD

# make the resource file
winaux.res : winaux.rc winaux.h
    rc -r winaux.rc

# make the object files
winaux.obj : winaux.c $(headers)
    $(cp)  -NT _WINAUXRS winaux.c

winauxfn.obj : winauxfn.c $(headers)
    $(cp) -NT _WINAUXRS winauxfn.c

winauxnt.obj : winauxnt.c $(headers)
    $(cp) -NT _WINAUXNT winauxnt.c

ttycls.obj : $(ttydir)\ttycls.c $(ttydir)\ttycls.h
    $(cp) -NT _WINAUXRS $(ttydir)\ttycls.c

# create the executable file, add resources, and make a map file
winaux.exe : winaux.def winaux.res \
             winaux.obj winauxfn.obj winauxnt.obj ttycls.obj
    link winaux+winauxfn+winauxnt+ttycls,/al:16,/m,$(LIBS),winaux
    rc winaux.res
    mapsym winaux
```

2

LISTING 3

```
; Definitions file for PMAUX

NAME

DESCRIPTION   'Presentation Manager Debugging Window'

STUB    'OS2STUB.EXE'

HEAPSIZE  4096
STACKSIZE 8192

CODE   MOVEABLE
DATA   MOVEABLE MULTIPLE

SEGMENTS
    _PMAUXRS PRELOAD MOVEABLE
    _PMAUXNT PRELOAD MOVEABLE DISCARDABLE

EXPORTS
    MainWndProc         @1
    AboutBoxProc        @2
```

LISTING 4

```
/* Resource file for the PM debugging window */

#include <pmwin.h>
#include "pmaux.h"

STRINGTABLE
{
    IDS_APPNAME   "PMaux"                 /* applicaton name */
    IDS_ICON      "PMOX"                  /* icon string */
    IDS_TITLE     "PM Debugging Window"   /* title bar */
}

MENU ID_RESOURCE
{
    SUBMENU "~Options",-1
    {
        MENUITEM "~CRonLF",IDM_CRONLF,MIS_TEXT | MIA_CHECKED
        MENUITEM SEPARATOR
        MENUITEM "~About...", IDM_ABOUT
    }
}
DLGTEMPLATE DT_ABOUT
{
    DIALOG "", 0,10,60,160,100, WS_SAVEBITS | FS_DLGBORDER
    {
        CTEXT "Microsoft Presentation Manager"  -1,4,88,156,8
        CTEXT "PMaux"                           -1,4,76,156,8
        CTEXT "Version 1.0, 11/30/88"           -1,4,64,156,8
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

3

```
        CTEXT "Created by"                      -1,4,52,156,8
        CTEXT "Bill Hall"                       -1,4,40,156,8
        CTEXT "3665 Benton Street, #66"         -1,4,32,156,8
        CTEXT "Santa Clara, CA 95051"           -1,4,24,156,8
        DEFPUSHBUTTON "~OK"     DID_OK, 64,4,32,14,WS_GROUP
    }
}
```

LISTING 5

```
/* Winaux resource file */
#include <style.h>
#include "winaux.h"

STRINGTABLE
BEGIN
    IDS_APPNAME,        "Winaux"
    IDS_WINTITLE,       "Winaux Debugging Window"
    IDS_ICONSTRING      "WAUX"
    IDS_HWND            "hWnd"  /* used in reading and writing WIN.INI */
    IDS_X               "x"
    IDS_Y               "y"
    IDS_CX              "cx"
    IDS_CY              "cy"
END

Winaux MENU
BEGIN
    POPUP "&Options"
        BEGIN
            MENUITEM "&CRonLF", IDM_CRONLF, Checked
            MENUITEM Separator
            MENUITEM "&About...", IDM_ABOUT
        END
END

DT_ABOUT dialog 22,17,154,95
STYLE WS_POPUP | WS_DLGFRAME
BEGIN
    CTEXT "Microsoft Windows"                   -1,0,5,154,8
    CTEXT "Winaux"                              -1,0,17,154,8
    CTEXT "Version 1.1, 11/30/88"               -1,0,29,154,8
    CTEXT "Created by"                          -1,0,39,154,8
    CTEXT "Bill Hall"                           -1,0,49,154,8
    CTEXT "3665 Benton Street, #66"             -1,0,57,154,8
    CTEXT "Santa Clara, CA 95051"               -1,0,65,154,8
    DEFPUSHBUTTON "&OK"                         IDOK,61,77,32,14,WS_GROUP
END
```

LISTING 6

```
/* Winaux main module */
#define NOCOMM
#include <windows.h>
#include <ttycls.h>

#define EXTERN
#include "winaux.h"

/* entry point for windows */
int FAR PASCAL WinMain(hInstance, hPrevInstance, lpszCmdLine, cmdShow)
HANDLE hInstance, hPrevInstance;
LPSTR lpszCmdLine;
int cmdShow;
{
    MSG msg;

    /* initialize program */
    if (!InitProgram(hInstance,hPrevInstance, lpszCmdLine, cmdShow))
      /* if we fail, then exit */
        return FALSE;

    /* otherwise read messages until program is terminated by user */
    while (GetMessage((LPMSG)&msg,NULL,0,0)) {
        TranslateMessage((LPMSG)&msg);
        DispatchMessage((LPMSG)&msg);
    }
```

4

```
    return msg.wParam;
}

/* main window procedure */
long FAR PASCAL MainWndProc(hWnd,message,wParam,lParam)
HWND hWnd;
unsigned message;
WORD wParam;
LONG lParam;
{
    PAINTSTRUCT ps;

    switch(message) {
      /* adjust position of text display if window is resized */
        case WM_SIZE:
            AdjustHeight(LOWORD(lParam), HIWORD(lParam));
            break;

      /* menu commands */
        case WM_COMMAND:
            WndCommand(hWnd, wParam);
            break;

      /* display string from remote program */
      case WM_USER:
            DispatchString(hWnd, wParam, lParam);
            break;

      /* initialize some things when window is created */
        case WM_CREATE:
            WndCreate(hWnd,lParam);
            break;

      /* quit */
        case WM_DESTROY:
            PostQuitMessage(0);
            break;

      /* set window handle to null in win.ini when closing application */
        case WM_CLOSE:
            SetWinIni(NULL);
            DestroyWindow(hWnd);
            break;

      /* if closing Windows, set window handle to null in win.ini */
        case WM_ENDSESSION:
            if (wParam)
                SetWinIni(NULL);
            break;

      /* refresh the window */
        case WM_PAINT:
            BeginPaint(hWnd, (LPPAINTSTRUCT)&ps);
            MainWndPaint(hWnd, (LPPAINTSTRUCT)&ps);
            EndPaint(hWnd, (LPPAINTSTRUCT)&ps);
            break;

      /* all other messages are handled by windows */
        default:
            return ((long)DefWindowProc(hWnd,message,wParam,lParam));
            break;
    }
    return(0L);
}
```

LISTING 7

```
/* PMAUX main module */
#define INCL_PM
#include <os2.h>
#include <stddef.h>
#include <string.h>

#include <ttycls.h>
#define EXTERN
#include "pmaux.h"
```

Code continues . . .

Hall Code, Continued

5

```
/* entry point for program */
int cdecl main(int argc, char *argv{})
{
    QMSG qmsg;

  /* call initialization */
    if (!InitProgram(argc, argv))
        return FALSE;

  /* fall into message loop */
    while (WinGetMsg(hAB, (PQMSG)&qmsg, (HWND)NULL, 0, 0))
        WinDispatchMsg( hAB, (PQMSG)&qmsg );

  /* program exit */
    WinDestroyWindow(hwndFrame);
    WinDestroyMsgQueue(hmqMsgQ);
    WinTerminate(hAB);
}

/* main window message loop */
MRESULT FAR PASCAL MainWndProc(HWND hWnd, USHORT msg, MPARAM mp1, MPARAM mp2)
{
    HPS hPS;            /* handle to PM display context */
    RECTL rect;         /* update rectangle */

    switch (msg) {
        case WM_COMMAND:        /* process menu commands */
            WndCommand(hWnd, LOUSHORT(mp1));
            break;

        case WM_USER:           /* process message from calling window */
            DispatchString(hWnd, mp1, mp2);
            break;

        case WM_CREATE:         /* called when main window is created */
            WndCreate(hWnd);
            break;

        case WM_CLOSE:          /* called when main window is closed */
            SetOS2Ini((HWND)NULL);
            WinPostMsg(hWnd, WM_QUIT, 0L, 0L) ;
            break;

        case WM_PAINT:          /* called when window needs repainting */
            hPS = WinBeginPaint(hWnd, (HPS)NULL, (PRECTL)&rect);
            MainWndPaint(hWnd, hPS, (short)rect.yTop, (short)rect.yBottom);
            WinEndPaint( hPS );
            break;

        case WM_ERASEBACKGROUND:        /* have PM take care of background */
            return(TRUE);
            break;

        default:                /* all other messages go here */
            return( WinDefWindowProc( hWnd, msg, mp1, mp2 ) );
            break;
    }
    return(0L);
}
```

LISTING 8

```
/* Extracts from the function support module for WINAUX */

/* adjust where text is to be displayed if window is resized */
void NEAR AdjustHeight(short width, short height)
{
    MWnd.Width = width;
    MWnd.Height = height;
    MWnd.Pos.y = height - MWnd.CHeight - 1;
    InvalidateRect(MWnd.hWnd, (LPRECT)NULL, TRUE);
}

/* process the menu */
void NEAR WndCommand(HWND hWnd, WORD wParam)
{
    HMENU hMenu = GetMenu(hWnd);
```

6

```
    FARPROC fp;
    HANDLE hInstance = GetWindowWord(hWnd, GWW_HINSTANCE);

    switch (wParam) {

        case IDM_ABOUT:
          /* make a proc instance for the about box window function */
            fp = MakeProcInstance((FARPROC)AboutBoxProc, hInstance);
          /* create a modal dialog box */
            DialogBox(hInstance, MAKEINTRESOURCE(DT_ABOUT),hWnd,fp);
            break;

        case IDM_CRONLF:
            MWnd.CRonLF = (MWnd.CRonLF ? FALSE : TRUE);
            CheckMenuItem(hMenu,wParam,MWnd.CRonLF ? MF_CHECKED : MF_UNCHECKED);
            break;
    }
}

/* save the passed string to a local buffer and display it */
void NEAR DispatchString(HWND hWnd, WORD wParam, LONG lParam)
{
    BYTE buf[100];
    int len, i;

    len = min((int)wParam, 100);
    for (i = 0; i < len; i++)
        buf[i] = *(LPSTR)(lParam + i);
   TTYDisplay((PTTYWND)GetWindowWord(hWnd,0),(short)len, buf);
}

/* set the window handle variable */
int SetWinIni(HWND hWnd)
{
    char buf[20];
    return (WriteProfileString((LPSTR)szAppName, (LPSTR)szhWnd,
                        (LPSTR)itoa(hWnd, buf, 10)));
}

/* This is the window proc for the about box */
BOOL FAR PASCAL AboutBoxProc(hDlg, message, wParam, lParam)
HWND hDlg;
unsigned message;
WORD wParam;
LONG lParam;
{
    switch (message) {

      /* nothing to initialize */
        case WM_INITDIALOG:
            break;

      /* this dialog box has only an OK button */
        case WM_COMMAND:
            switch (wParam) {
              case IDOK:
              /* destroy the dialog box */
                  EndDialog(hDlg,TRUE);
                  break;
              default:
                  return FALSE;                 /* we did not process */
            }
            break;
        default:
            return FALSE;
    }
    return TRUE;                /* we processed message */
}
```

LISTING 9

```
/* Extracts from the function support module for PMAUX */

/* respond to menu commands */
void NEAR WndCommand(HWND hWnd, USHORT wParam)
{
    HWND hMenu;
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
    switch (wParam) {
          case IDM_ABOUT:          /* display about box */
              WinDlgBox(HWND_DESKTOP, hWnd, AboutBoxProc, NULL, DT_ABOUT, NULL);
              break;

          case IDM_CRONLF:
              MWnd.CRonLF = (MWnd.CRonLF ? FALSE : TRUE); /* toggle parameter */
              hMenu = WinWindowFromID(hwndFrame, FID_MENU);  /* get menu */
            /* set or remove check mark on menu */
              WinSendMsg(hMenu, MM_SETITEMATTR, MPFROM2SHORT(wParam, TRUE),
                         MPFROM2SHORT(MIA_CHECKED, MWnd.CRonLF ? MIA_CHECKED : 0));
              break;
    }
}

/* send the received buffer to the display */
void NEAR DispatchString(HWND hWnd, MPARAM mp1, MPARAM mp2)
{
#define LBUFLEN 80

    BYTE buf[LBUFLEN];
    int len, i, count;
    BYTE FAR *bufptr;
    SEL sel;

    len = (int)LOUSHORT(mp1);    /* read the length */
    sel = (SEL)LOUSHORT(mp2);    /* get the selector of the shared buffer */
    bufptr = MAKEP(sel,0);       /* make it into a pointer */

    if (DosGetSeg(sel) == 0) {  /* access the shared buffer */
        while (len > 0) { /* read and dispatch each part of buffer until done
            count = min(len, LBUFLEN);
            for (i = 0; i < count; i++)
                buf[i] = *bufptr++;
            TTYDisplay(&MWnd, (short)count, buf);
            len -= count;
        }
    }
    else        /* something wrong, ring the bell */
        WinAlarm(HWND_DESKTOP, WA_WARNING);
}

/* set the window handle (might be NULL) into OS2.ini */
BOOL SetOS2Ini(HWND hWnd)
{
    char buf[30];
   return(WinWriteProfileString(hAB,szAppName,"hWnd",ltoa((LONG)hWnd ,buf,10)));
}

/* simple procedure to handle the about box */
MRESULT EXPENTRY AboutBoxProc(HWND hDlg, USHORT msg, MPARAM mp1, MPARAM mp2)
{
    switch (msg) {
        case WM_COMMAND:
            switch(COMMANDMSG(&msg)->cmd) {
                case DID_OK:
                    WinDismissDlg (hDlg, TRUE);
                    break;
                default:
                    break;
            }
            break;
        default:
            return WinDefDlgProc(hDlg, msg, mp1, mp2);
    }
    return FALSE;
}
```

LISTING 10

```
/* extracts from WPMAUXNT.C */

/* enter here to initialize the program */
BOOL FAR InitProgram(hInstance,hPrevInstance, lpszCmdLine, cmdShow)
HANDLE hInstance, hPrevInstance;
LPSTR lpszCmdLine;
```



```
int cmdShow;
{
  /* if first instance, do these things */
    if (hPrevInstance == NULL) {
       LoadString(hInstance,IDS_ICONSTRING,(LPSTR)szIconTitle,
                                    sizeof(szIconTitle));
       LoadString(hInstance,IDS_HWND,(LPSTR)szhWnd,sizeof(szhWnd));
        if (!RegisterWindowClass(hInstance))
            return FALSE;
    }
    else
        return FALSE;
  /* no additional instances allowed */

  /* read the rectangle for create window */
    getcrect(hInstance);

  /* create the window */
    if (!MakeAndShowMainWnd(hInstance,cmdShow))
        return FALSE;

  /* write the handle to win.ini */
    if (!SetWinIni(MWnd.hWnd))
        return FALSE;

  /* show success */
    return TRUE;
}

/* register the windows class */
static BOOL near RegisterWindowClass(hInstance)
HANDLE hInstance;
{
    PWNDCLASS pWndClass;
    HANDLE hTemp;

    /* standard stuff except for... */
    pWndClass->cbWndExtra = sizeof(PTTYWND);    /* space for ptr to MWnd */

    /* more standard stuff */
}

/* get the user's window size or load and write default values */
static void NEAR getcrect(HANDLE hInstance)
{
    char buf[20];
    char keystr[20];
    short temp;

  /* get the key string from resources */
   LoadString(hInstance,IDS_X,(LPSTR)keystr,sizeof(keystr));
  /* try to read the value */
    temp = GetProfileInt((LPSTR)szAppName, (LPSTR)keystr, -1);
  /* value is negative, so use a default position */
    if (temp <= 0) {
        crect.left = CW_USEDEFAULT;
        if (temp < 0)
            WriteProfileString((LPSTR)szAppName, (LPSTR)keystr,
                                (LPSTR)itoa(crect.left, buf, 10));
    }
  /* we found one, so use this below when creating the window */
    else
        crect.left = temp;

  /* repeat for the y position, x extent, and y extent */
}

/* create a window of the class registered above */
static BOOL NEAR MakeAndShowMainWnd(hInstance, cmdShow)
HANDLE hInstance;
int cmdShow;
{
    char szWinTitle[30];

  /* get the text for the title */
```

Code continues . . .

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Hall Code, Continued

9

```
    LoadString(hInstance, IDS_WINTITLE, (LPSTR)szWinTitle,sizeof(szWinTitle));

  /* create the window using the rectangle calculated above */
    MWnd.hWnd = CreateWindow((LPSTR)szAppName,  /* class */
                             (LPSTR)szWinTitle, /* title string */
                             WS_OVERLAPPEDWINDOW,       /* style */
                             crect.left,   /* values obtained from getcrect */
                             crect.top,
                             crect.right,
                             crect.bottom,
                             (HWND)NULL,        /* no parent */
                             (HMENU)NULL,       /* class menu */
                             (HANDLE)hInstance, /* instance */
                             (LPSTR)&MWnd);     /* ptr to window structure
                                                   will be retrieved
                                                   at WM_CREATE time */

  /* successful if BOTH the window and the text buffer are created */
    if (MWnd.hWnd && MWnd.hVidBuf) {
        ShowWindow(MWnd.hWnd, cmdShow);
        UpdateWindow(MWnd.hWnd);
        return TRUE;
    }
  /* return failure */
    return FALSE;
}

/* this routine is called upon receipt of the WM_CREATE message */
void WndCreate(hWnd, lParam)
HWND hWnd;
LONG lParam;
{
    HDC hDC;
    TEXTMETRIC TM;
    short width, height, cwidth, cheight;
    LPCREATESTRUCT pCS;
    PTTYWND pMWnd;

  /* we are going to create a buffer for a window this large */
    width = GetSystemMetrics(SM_CXFULLSCREEN);
    height = GetSystemMetrics(SM_CYFULLSCREEN);

  /* get the font size */
    hDC = GetDC(hWnd);
    GetTextMetrics(hDC, &TM);
    cwidth = TM.tmAveCharWidth;
    cheight = TM.tmHeight + TM.tmExternalLeading;
    ReleaseDC(hWnd, hDC);

  /* here we retrieve the pointer to our local window stucture */
    pCS = (LPCREATESTRUCT)lParam;
    pMWnd = (PTTYWND)LOWORD(pCS->lpCreateParams);

  /* initialize the extra data to the pointer to the TWnd structure */
    SetWindowWord(hWnd,0,(WORD)pMWnd);

  /* now create the text buffer and set some defaults */
   InitTTYWindow(pMWnd,0,0,width,height,cwidth,cheight,FALSE,TRUE,TR UE,0x7f);
}
```

LISTING 11

```
/* TTY window header file */

#ifdef WINDOWS
#define HPS HDC         /* this saves some ifdefs in the code */
#endif

struct TTYWND {
    HWND hWnd;          /* window handle */
#ifdef WINDOWS
    POINT Pos;          /* a pair of shorts in Windows */
    HANDLE hVidBuf;     /* handle to heap area containing buffer */
#else
    POINTL Pos;         /* a pair of longs in PM */
#endif
    BYTE *pVidBuf;              /* pointer to text buffer in heap */
```

10

```
    short CWidth, CHeight;      /* character width and height of font */
    short Top, Left;            /* current location of window */
    short Width, Height;        /* current size (or logical size) */
    short MaxLines,MaxCols;     /* maximum size */
    short MaxLineLength;        /* maxcols + 1 */
    short CurLineOffset;        /* offset in current line of character */
    short oCurrentLine;         /* offset in buffer to current line */
    short oVidLastLine;         /* offset to last line in buffer */
    BOOL LFonCR;                /* do lf on cr if TRUE */
    BOOL CRonLF;                /* do cr on lf if TRUE */
    BOOL Wrap;                  /* wrap lines if TRUE */
    BYTE ebitmask;              /* character mask - 0xff or 0x7f */
};
typedef struct TTYWND *PTTYWND;

/* shared function declarations */
InitTTYWindow(PTTYWND,short,short,short,short,short,short,BOOL,BO OL,BOOL,BYTE);
int NEAR TTYDisplay(PTTYWND pTTYWnd, short len, BYTE *str);
void NEAR TTYWndPaint(PTTYWND pTTYWnd, HPS hPS, short top, short bottom);
```

LISTING 12

```
/* TTYCLS module - excerpts
 *
 * This file supports a teletype-style window in either Microsoft
 * Windows or Presentation Manager.
 *
 * If using in a Windows program, be sure to define WINDOWS.
 */

/* Called at create window time.  Creates a text buffer based
 * on the passed-in width and height and the character width and height.
 * Other parameters, such as wrap, character mask, etc., are also set.
 */
InitTTYWindow(pTTYWnd,left,top,width,height,charwidth,charheight, LFonCR,CRonLF,
                        wrap,mask)
PTTYWND pTTYWnd;
short left, top, width, height, charwidth, charheight;
BOOL LFonCR, CRonLF, wrap;
BYTE mask;
{
    int bufsize;

    pTTYWnd->Left = left;
    pTTYWnd->Top = top;
    pTTYWnd->Width = width;
    pTTYWnd->Height = height;
    pTTYWnd->CWidth = charwidth;
    pTTYWnd->CHeight = charheight;
    pTTYWnd->MaxLines = height / charheight;
    pTTYWnd->MaxCols = width / charwidth;
    pTTYWnd->MaxLineLength = pTTYWnd->MaxCols + 1;
    bufsize = pTTYWnd->MaxLines * pTTYWnd->MaxLineLength;

  /* allocate space for the display characters */
  /* if allocation successful, initialize the remaining tty data */

#ifdef WINDOWS
    if (pTTYWnd->hVidBuf = LocalAlloc(LMEM_FIXED | LMEM_ZEROINIT, bufsize)) {
        pTTYWnd->pVidBuf = LocalLock(pTTYWnd->hVidBuf); /* lock the buffer */
        pTTYWnd->Pos.y = (pTTYWnd->MaxLines - 1) * charheight - 1;
#else
    if (pTTYWnd->pVidBuf = WinAllocMem(hHeap, bufsize)) {
        memset(pTTYWnd->pVidBuf, NUL, bufsize);
        pTTYWnd->Pos.y = 1;
#endif
        pTTYWnd->Pos.x = 0;
        pTTYWnd->oCurrentLine = pTTYWnd->CurLineOffset = 0;
        pTTYWnd->oVidLastLine = (pTTYWnd->MaxLines-1) * pTTYWnd->MaxLineLength;
        pTTYWnd->LFonCR = LFonCR;
        pTTYWnd->CRonLF = CRonLF;
        pTTYWnd->Wrap = wrap;
        pTTYWnd->ebitmask = mask;
      return TRUE;
    }
    return FALSE;
}
```


PRODUCED 2005 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
/* display the characters contained in str */
int NEAR TTYDisplay(PTTYWND pTTYWnd, short len, BYTE *str)
{
    HPS hPS;
    register BYTE *ptr;
    register short ctr;
    short toff, txpos;
    BYTE *tbuf;

    short cols = pTTYWnd->MaxCols;
    short cwidth = pTTYWnd->CWidth;
    BYTE mask = pTTYWnd->ebitmask;

    while (len) {
        ptr = str;
        ctr = 0;
        txpos = (short)pTTYWnd->Pos.x;
        tbuf = pTTYWnd->pVidBuf + pTTYWnd->oCurrentLine;
        toff = pTTYWnd->CurLineOffset;

        while((*ptr &= mask) >= SP) {
            if ((len) && (toff < cols)) {
                ctr += 1;
                *(tbuf + toff++) = *ptr++;
                txpos += cwidth;
                len -= 1;
            }
            else
                break;
        }
        if (ctr) {
#ifdef WINDOWS
            hPS = GetDC(pTTYWnd->hWnd);
            TextOut(hPS, pTTYWnd->Pos.x, pTTYWnd->Pos.y, (LPSTR)str, ctr);
            ReleaseDC(pTTYWnd->hWnd, hPS);
#else
            hPS = WinGetPS(pTTYWnd->hWnd);
            GpiSetBackMix(hPS, BM_OVERPAINT);
           GpiCharStringAt(hPS,(PPOINTL)&pTTYWnd->Pos,(LONG)ctr,(PCH)str);
            GpiRestorePS(hPS, -1L);
            WinReleasePS (hPS);
#endif
            if (toff < cols) {
                pTTYWnd->CurLineOffset = toff;
#ifdef WINDOWS
                pTTYWnd->Pos.x = txpos;
#else
                pTTYWnd->Pos.x = (LONG)txpos;
#endif
            }
            else
                if (pTTYWnd->Wrap) {
                    DoCR(pTTYWnd);
                    DoLF(pTTYWnd);
                }
        }
        while ((*ptr &= mask) < SP) {
            if (len) {
                switch(*ptr) {
                    case BEL:
#ifdef WINDOWS
                        MessageBeep(0);
#else
                        WinAlarm(HWND_DESKTOP, WA_ERROR);
#endif
                        break;
                    case HT:
                        DoTab(pTTYWnd);
                        break;
                    case CR:
                        DoCR(pTTYWnd);
                        if (pTTYWnd->LFonCR)
                            DoLF(pTTYWnd);
                        break;
```



```
                    case LF:
                        if (pTTYWnd->CRonLF)
                            DoCR(pTTYWnd);
                        DoLF(pTTYWnd);
                        break;
                    case BS:
                        DoBS(pTTYWnd);
                        break;
                }
                len -= 1;
                ptr++;
            }
            else
                break;
        }
        str = ptr;
    }
    return (len);
}

/* repaint the window */
void NEAR TTYWndPaint(PTTYWND pTTYWnd, HPS hPS, short top, short bottom)
{
#ifdef WINDOWS
    POINT pt;
    short wtop;
#else
    POINTL pt;
#endif
    BYTE *lineptr;
    BYTE *pBuf, *pEnd;
    short lines, length;
    short cheight;
    int i;

#ifndef WINDOWS
    GpiErase (hPS);
#endif
    pt.x = 0;
    pBuf = pTTYWnd->pVidBuf;
    pEnd = pBuf + pTTYWnd->oVidLastLine;

    lineptr = pBuf + pTTYWnd->oCurrentLine;
    pt.y = pTTYWnd->Pos.y;

    length = pTTYWnd->MaxLineLength;
    cheight = pTTYWnd->CHeight;
#ifdef WINDOWS
    wtop = pTTYWnd->Height - top;
    lines = wtop / cheight;
    if (wtop % cheight)
        lines += 1;
    lines = min(pTTYWnd->MaxLines, lines);
#else
    lines = top / cheight;
    if (top % cheight)
        lines += 1;
    lines = min(pTTYWnd->MaxLines, lines);
#endif

    for (i = 0; i < lines; i++) {
#ifdef WINDOWS
        TextOut(hPS, pt.x, pt.y, (LPSTR)lineptr, strlen(lineptr));
        pt.y -= cheight;
#else
       GpiCharStringAt(hPS,&pt,(LONG)strlen(lineptr),(PCH)lineptr);
        pt.y += (LONG)cheight;
#endif
        if ((lineptr -= length) < pBuf)
            lineptr = pEnd;
    }
}
```


PRODUCED BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED
©2004 BY UNZ.ORG

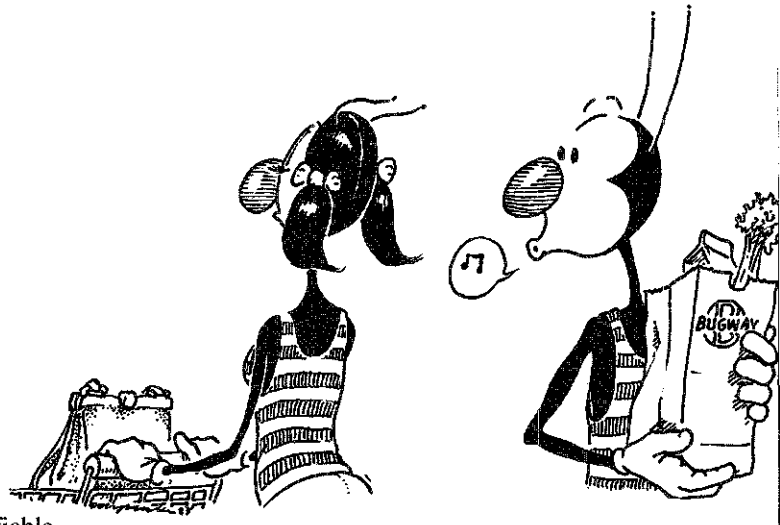
The Proposed ANSI C Language Standard

by
Rex Jaeschke

© *Rex Jaeschke, 1989. All rights reserved.*

In each of the recent installments, I've reviewed a C compiler's support for the ANSI C standard. Beginning in this article, I will be adding some information that is not compiler specific. That way, if the product being reviewed is not of interest to you, the column should still provide some useful information.

Recently, while working on my portability software suite, I came across a couple of situations that were poorly handled by all or most of my DOS compilers. The problems have to do with the type qualifier **const** and structure expressions.



-STRUCTURE EXPRESSION-

const Is Not Always Constant

The primary reason for adding the **const** type qualifier to ANSI C was to allow the compiler to diagnose an attempt to modify a memory location flagged as nonmodifiable. The **const** keyword can be applied to the basic arithmetic and pointer types, and all compilers I have tested in this manner enforce the required checking. However, when used with pointers in a function prototype, only two out of the six compilers tested (Microsoft and Turbo) complained properly—and then only as warnings that would otherwise not have been seen if that level were disabled. (The six compilers tested were the latest version of Microsoft, Turbo, Watcom, Lattice, MIX, and DeSmet.) Here is the code in question:

```
/* passing a ptr to const char is not assignment-com-
patible with the ptr to non-const char expected by the
prototype. Therefore, should not accept the function call
as written.
*/
#include <stdio.h>

void f(char *);
void main()
{
        const char c = 'A';
        const char *pc = &c;

        printf("c = %c\n", c);
        f(pc);          /* should not compile */
        printf("c = %c\n", c);
}

void f(char *pc)
{
        *pc = 'X';
}
```

The purpose of a prototype is to make sure the number of arguments in the function call matches that indicated by the declaration and to ensure that each actual argument is assignment compatible with that expected. That is, calling a function in the presence of a prototype is akin to assigning the actual arguments to objects of the type specified in the prototype argument list. In the case of assignment, the following is disallowed:

```
f()
{
        const int *pci;
        int *pi;

        pi = pci;       /* error */
}
```

A pointer to a **const** object cannot be assigned to a pointer to a non-**const** object, since the **const** protection for that location would be lost.

A similar situation exists in the earlier example, where a pointer to **const char** actual argument is being "assigned" to a pointer to non-**const char** formal argument. The two are *not* assignment compatible and, under ANSI rules, must be flagged as an error, not as a warning. (Note that you can assign a pointer to a non-**const** object to a pointer to a **const** object; it merely adds more access protection and is always safe.)

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Interesting Structure Expressions

Consider the following program:

```
struct s {
        char m[30];
};

struct s s1;
struct s s2 = {"abcdef"};

main()
{
        struct s frs(void);
        char *pc;

/*1*/   pc = (s1, s2).m;
/*2*/   pc = frs().m;
/*3*/   pc = (5 ? s2 : frs()).m;
/*4*/   pc = (s1 = s2).m;
}
```

Cases 1 through 4 caused three of the compilers to produce errors, two to hang up my system, and one to produce an internal compiler error message. All four cases are correct, at least under ANSI rules. I believe they should be correct even prior to ANSI, although arguing that is very subjective, since K&R doesn't tell you all you need to know about this and other areas.

Most compilers complained that an address was needed and that since the comma, **?**, function call, and assignment operators do not generate lvalues (by definition), the results they produce cannot have their address taken. How then could I expect a member offset to be accessed if the address of the parent structure could not be found? That's a reasonable question, but let's look at the wording in the standard:

"The first operand of the dot operator shall have a qualified or unqualified structure or union type, and the second operand shall name a member of that type." Certainly, all four cases comply with this.

"A postfix expression followed by a dot [operator] and an identifier designates a member of a structure or union object. The value is that of the named member, and is an lvalue if the first expression is an lvalue." Since none of the first operands in our four cases is an lvalue, none of the member references is an lvalue—and that's fine, because we never try to use them in a context requiring an lvalue.

From an overt programming point of view, none of these cases can be justified. However, cases 2 and 3 do have some merit, since to rewrite them without the embedded function call would require the result of the function to be copied to a temporary structure so that it could be referenced later with the dot operator. Since structure assignment can be quite expensive on some implementations, this approach may not be desirable. Of course, you can always argue that assigning, passing, and returning structures by value can almost always be avoided (by using pointers to structures instead), thus eliminating strange expressions such as those shown above and the corresponding failure of implementers to handle these "fringe" cases.

MIX Power C V1.2

Pst! Hey, Mac. Wanna buy a cheap K&R C compiler with symbolic debugger and library source for less than 100 bucks? And just for you, I'll throw in a packed-decimal math library and a 650-page manual. Wadda ya say, huh?

Does it sound too good to be true? Well, such a product is alive and well and, after an initial spurt of sales and promotion activity several years ago, it's back on the scene in force. The product is Power C, from MIX Software in Richardson, Texas.

Before I get into the results of my gruelling ANSI conformance/portability software suite tests, a few general comments are in order. The product is quite well engineered and the documentation is good, although personally I don't care for bound manuals. All the K&R stuff is there, including a comprehensive run-time library.

The Compiler Options

You can select either signed or unsigned "plain" chars using **/j**. Definitely don't use **/n**, which permits nested comments. The recommended way to comment out code that contains comments is to use **#if 0/#endif**. The **/x** option shows macro expansions in the listing file.

The Preprocessor

The following ANSI C attributes are not supported or had some limitations:

- No **##** operator.
- Permits the predefined macros and the defined operator to be **#undef**ed.
- Incorrectly believes the **#** operator can exist in macros without arguments.
- Macro definitions containing their own names are recursive, hanging up the system.
- The format of **__DATE__** is not quite correct, although its value is.
- Embedded white space in macro arguments doesn't always get handled correctly when used with the **#** operator.
- Should not permit a macro called **defined** to be defined.
- Does not diagnose extra unneeded tokens at the end of directive lines.
- Should not permit part of **#if/#endif** directive groups to be in different source files.
- Should not permit macros to expand into directive names that are then interpreted.
- Should not recognize **enum** constants, floating-point constants, and casts in **#if** arithmetic.
- **#if 10/0** produced an answer of 1 with no warning.
- Won't permit **!** with defined operator in **#if**.
- **#error** is not implemented correctly.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

The **sizeof** operator is permitted in **#if** arithmetic, although this is not required by ANSI C.

Many of my library tests use **#undef** with names and then take the address of those names to make sure a real function exists in the library. This caused great confusion, however. For example, after having done **#undef sqrt**, the compiler complained that **sqrt** was not known, even though math.h was included and **sqrt** was a function. It appears that **#undef** removes symbols other than macro names from the identifier table as well—and, of course, it shouldn't.

According to the release notes, some **#pragma** directives are supported. One big plus in this area is MIX's issuing of a warning message when it comes across a pragma it does not recognize. While this is not required, it is the only way you will know if pragmas are being misinterpreted when you port in code that contains pragmas.

The format of the listing file showing the "before" and "after" versions of source code is very useful. It is easy to see how macros have expanded.

The C Language

The following ANSI features are not supported or have problems:

- The general backslash token continuation idea is only partially implemented.
- No union initialization.
- **const** is recognized but is given no semantics; so it is ignored.

All other ANSI requirements seem to be supported. These include string concatenation, prototypes, trigraphs, bit fields in unions, auto aggregate initializers, **const**, volatile and signed keywords, float and long double constants, and void pointers.

The Run-Time Library

Any header not discussed here exists, unless specified otherwise.

- The assert macro in assert.h needs some work since, like many implementations, it expands to an if statement instead of a void expression.
- The unary minus signs in float.h seem to cause considerable grief by hanging up the system. (Unary minus is an ANSI invention.)
- The multibyte macro is missing from limits.h.
- The ANSI-invented locale.h is provided, although the macros and structure template are missing.
- The math library declared in math.h includes some extensions that are enabled in ANSI mode and should not be.
- The type **sig_atomic_t** in signal.h must be a typedef, not a macro.
- The **va_arg** macro in stdarg.h involves applying the ++ operator to the result of a cast expression. In ANSI C, a cast does not generate an lvalue and therefore cannot be used with the ++ or -- operator. While this can be an extension, it must not be used in ANSI mode.
- In stddef.h (and other headers), **size_t** is a macro instead of a typedef. Also, the **offsetof** macro definition is really broken, and it clearly could not have been tested properly even once. Not only are some casts missing, but the expansion relies on one address being subtracted from another. While that is permissible, it does not constitute a compile-time constant expression, as required by this macro. **errno** is declared here, and it should not be. (That is a carry-over from an early ANSI C draft.)
- The two macros last added to stdio.h—**FOPEN_MAX** and **FILENAME_MAX**—are missing.
- The **EXIT_*** macros are missing from stdlib.h, along with NULL, the multibyte macro, and the type **wchar_t**. Also yet to come are the multibyte conversion functions. (I have yet to see any library with these.)
- NULL is missing from string.h.
- The predefined macro **__STDC__** is defined to be 1, indicating the implementation is ANSI conforming. However, this claim is premature. Also, some parts of some headers are conditionally compiled based on a macro called **ANSI**. Since these two macros should mean the same thing, I would expect the use of **ANSI** to be discontinued in the future.
- The identifiers in function prototypes all need their names changed so that they begin with two underscores or an underscore and a capital letter, to avoid infringing on programmer name space. Also, the **const** keyword is missing from the many places it should exist in the standard library prototypes.
- Numerous headers conditionally compile typedefs and structure template declarations (time.h is a good example); however, they do so based on macro names that belong to programmers, not implementers. Also, if you include these same headers multiple times, you will get errors; you should not, since the controlling macros that are tested for never get defined.

The Bottom Line

Despite the large number of preprocessor and library shortcomings listed here, the MIX compiler passed far more tests than it failed, and it ranks above average with regards to supporting the emerging ANSI C Standard. In fact, it is quite well placed, considering that many of the library problems can be fixed simply by editing the headers.

All in all, the MIX package provides a lot of speed and capability for the price, and the documentation is decent. Despite its rough edges, it gives value for the money.

Rex Jaeschke is an independent computer consultant, author, and lecturer. He is also a member of the X3J11 C Standards Committee, observer on the ISO C Standards Committee, founding editor of The C Journal, *and C language editor of* DEC PROFESSIONAL. *His new book,* Portability and the C Language, *was published by Hayden Books in October 1988. It is the first text on this popular topic.*

If you would like to comment on, or question any aspect of, the ANSI C standard, write to Rex at 2051 Swans Neck Way, Reston, VA 22091, or contact him via UUCP at uunet!aussie!rex. ♦

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

OMNIVIEW
ROCK SOLID MULTITASKING!

Packed with the power you need, OMNIVIEW takes you beyond the limits of DOS.

- Time-slices multiple jobs
- Takes full advantage of EMS 4.0
- Optional support for '386 Systems
- Manages megabytes of programs
- Controls resident utilities
- Lets you cut and paste between programs

OMNIVIEW unleashes your PC & converts it into an easy-to-use, multitasking workstation. Whether you have an 80386 or 8088 based computer, OMNIVIEW will squeeze out every ounce of power.

Solid & bulletproof, it's 47K assembly language engine gives you all the power without added overhead or complexity. Just ask the reviewers.*

"Sunny Hill provides true multitasking for most PC applications. It performs remarkably well, causing almost no noticeable reduction in the performance of each program . . . The user manual is well organzied and easy to read . . . It is an excellent product."
PC Tech Journal Sept '87

"It does an amazing job of handling not only multitasking, but in many cases, concurrent processing as well. I was amazed at it's success in handling the "impossible" combinations I threw at it."
"It is one of the best designed & implemented programs I've ever worked with. It is also one of the most bulletproof."
"What sets it apart from the pack is the fact that it runs in a network environment."
Data Based Advisor Aug '87

OMNIVIEW requires an IBM PC, XT, AT or Jr compatible, and PC or MS DOS 2.x to 3.x. To get your copy, call toll free:
(800) 367-0651
or send $89.95 + 3.00 S&H ($8.00 Intl.) to:

Sunny Hill Software
PO Box 55278, Seattle, WA 98155-5278

For more information call (206) 367-0650 *Omniview formerly Taskview
DoubleDOS trademark Softlogic Solutions, Concurrent PC-DOS reg. trademark Digital Research Corp., MS Windows trademark Microsoft Corp., DESQview trademark Quarterdeck Office Systems, Inc. TopView trademark IBM Corp.

Number 85 on the Reader Service Card

CHANGE OF ADDRESS?

Please send us your new address along with your previous ZIP code, using this handy form— Thanks.

NAME _____
COMPANY _____
ADDRESS _____
CITY _____ STATE _____
ZIP CODE _____ OLD ZIP CODE— _____
TELEPHONE _____

Please mail to:
Programmer's Journal
P. O. Box 30160
Eugene, OR 97403

Or call toll free
800-234-0386

DOS & OS/2
Free Demo Diskette Available

Q. How many programmers does it take to maintain a MAKE dependency file?

A. NONE! *If you use* VersiMAKE™

VersiMAKE™ is a full-featured MAKE utility that includes:

- **Dependency Generation**
  Derives your system's dependencies, through analysis of its C and MASM source files. Say goodbye to manual maintenance of MAKE dependency files!
- **Wild Card File Name Matching**
  Analyzes an entire collection of source files with a single statement.
- **Nested Include Files**
  Handles standard C and MASM "include" conventions, and the INCLUDE environment variable.
- **Powerful Macro Facilities**
  Built-in macros, user-defined macros, and environment variables.
- **Analytical Reports**
  Shows the entire Include file hierarchy for each source file analyzed, and all of the parent source files for each Include file.

Q. How many programmers does it take to trace a symbol thru your system?

A. ONE! *If you use* VersiCREF™

VersiCREF™ is a unique utility that creates a Master Cross-Reference of your entire system.

- **Multi-Lingual**
  Handles C, assembler, or both.
- **Flexible**
  File names with line numbers, or file names alone. Global and local symbols, or *globals alone.*
- **Powerful**
  Easily handles systems containing 100 source files or more.

VersiMAKE™	$125
VersiCREF™	$75
Both	$150

(Outside U.S., add $5 for shipping and handling)

800-334-4096
(In NJ, 609-871-0202)
MC/VISA/AMEX accepted

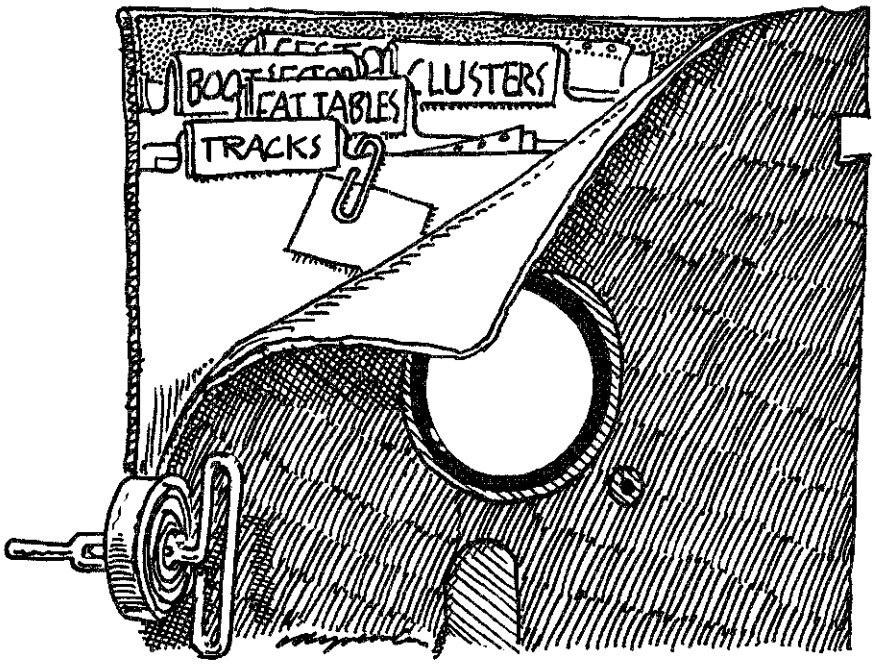
SUMMIT INFORMATION SYSTEMS, INC.
73 East Lane, Willingboro, NJ 08046

Number 158 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Disk Formatting

by
John Newlin



System-level programming on the PC can be both fun and challenging. It's fun because you are usually operating at least at the DOS level and frequently at the BIOS level. It's challenging because it can be real tricky, particularly in the face of conflicting documentation—or none at all—and debugging can be difficult. But I can think of no more rewarding activity for a programmer than to design and build a systems-level utility.

This article will focus on building just such a utility: a floppy disk formatter. A few years ago—without the current proliferation of differing disk types, formats, drives, and controllers—coding a formatting utility for the PC was a much more straightforward task than it is today, Even so, it is still possible to build a formatter that will format all available disk types in most machines. Of course, it can be argued that since every DOS comes with a FORMAT program, there is no pressing operational need for such a utility. Maybe not. But creating one is an excellent exercise in systems-level programming, and a creative, clever programmer might make the result much more utilitarian than the DOS-provided formatter.

Before getting into the design and coding of this utility, let's discuss some information about disk formats and drives. The 5.25", 1.2M, dual-capacity drive made its debut with the IBM AT-class machine. It is designed to format, read, and write to either a 9-sector, 40-track, 360K disk or a 15-sector, 80-track, 1.2M disk. The 3.5", 1.44M, dual-capacity drive supports either a 9-sector, 80-track, 720K disk or an 18-sector, 80-track, 1.44M disk. In both cases, the disks used for high-capacity storage are specifically manufactured to accommodate the greater amount of data to be stored in the same physical area. These disks are, accordingly, more expensive.

The dual-capacity drive is capable of supporting both low- and high-data capacity by varying two physical performance parameters: the head step rate and the rate at which the data is transferred to the disk. When accessing a 5.25" low-capacity disk, this drive will be in what's called a double-step mode, and the transfer data rate will be set to 300 kilobits per second. When accessing a 1.2M disk, the data rate is reset to 500 kilobits per second, and the head single-steps. The head-step-rate change accommodates the doubling of tracks on the 1.2M disk and is one of the primary reasons for problems in non–dual-capacity drives reading disks formatted and written to by high-capacity drives. The 3.5" high-capacity drive needs only to increase the data transfer rate to store twice the data on each track. In both cases, the disk controller must be informed of type of media—high or low capacity—before formatting; otherwise, there is no way for the drive or controller to know whether the disk can handle the high data rate.

The first step in the disk formatting project is the same as in any other programming project: Analyze the problem and determine what, if any, system-level resources are available. An examination of the DOS and BIOS level technical reference documentation reveals that most of the system interface will have to be accomplished through ROM BIOS functions. In fact, there is a specific interrupt service in the disk interface section (INT 13H) of the BIOS that will format a disk track.

INT 13H Disk Interface

A closer look at function 05H, **Format Desired Track,** raises some interesting questions, however. There's nothing in the listing that refers to the different disk formats. Will this call automatically adjust for a 1.2M or a 720K drive? How, if at all, can the BIOS know whether the user has placed a high-capacity disk in a dual-capacity drive? Hmmmm . . . looks

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

like we've got some more homework to accomplish before plunging into the code.

Further examination of the disk–BIOS interface reveals several other function calls that will obviously be needed if our formatter is to handle different disk formats. For example, function 08H, **Read Drive Parameters,** returns a wealth of useful information, and it turns out that there are two functions that will set up the BIOS for a specific type of disk medium. Either function 17H, **Set DASD Type for Format,** or function 18H, **Set Media Type for Format,** may be used in the case of a dual-capacity drive, and at least one must be called prior to invoking function 05H. Two other functions will be useful in a support role for the formatter: function 03H, **Write Desired Sectors from Memory**, and function 04H, **Verify Desired Sectors.**

Now that we've identified the essential system interfaces, let's look more closely at function 05H. There are two data tables related to the use of this function, one direct and one indirect. The programmer must build the directly related table before making the BIOS call. The second table already resides in ROM, but it must be located and moved to a DOS area in RAM.

The table that must be built is called the *track address field table*. It's constructed of four-byte entries, one for each sector to be placed on a disk track, and the structure of each entry looks like this Turbo Pascal record:

```
AddressFieldType = record
                     TrackNum,HeadNum,SectNum,SizeCode
                                 : byte;
                     End;
```

The first three entries are for the track, head (or side), and sector numbers. The last is a code that specifies the number of bytes per sector. Unless you desire to format disks nobody else can read, you should use code = 2 for 512 bytes. Since there must be one such record for each sector on a track and the maximum number of sectors likely to be encountered is 18 (for a 1.44M disk), the table array should be defined as follows:

```
Const
  MaxSectors = 18;

Var
  AddressFields : array[1..MaxSectors] of AddressFieldType;
```

Loading this table and updating it during the progress of the format will be covered later.

The other table of concern is the *disk format parameter table*, which is of prime importance in achieving a successful format. It is an eleven-byte table, and the ROM BIOS contains one for each disk type supported. Thus, in a machine equipped with dual-capacity, 5-1/4", 1.2M, and 3-1/2", 1.44M drives, the ROM BIOS will contain four such tables, one for each low-capacity (360K and 720K) format and one for each high-capacity (1.2M and 1.44M) format. When the machine is booted, DOS downloads one of these tables, usually the one supporting the maximum disk capacity in drive A, to a work area in RAM. Our job will be to locate the proper table and to move its values to the RAM work area that is accessed by the BIOS during formatting.

First we have to find the address of the work area. Normally, it resides at 0000H:0522H, but to be safe we can retrieve its location from the system interrupt table in low RAM. Interrupt 1EH is not used as a normal interrupt. Instead, it points to the area in RAM where the format parameter data is stored. This table structure is defined in the following manner:

```
Type
  DiskParameterTableType = Record
                             StepRate, LoadTime, Shutoff,
                             BytesSec, SecTrack, GapLen,
                             DataLen, FGapLen, FillByte,
                             SettleTime,StartTime : byte;
                           End;
```

If we do things correctly, there will be no need to alter any of these values. The only task will be to find the correct data in ROM for the target disk and load the RAM table with that data. The location of the desired table in ROM can be obtained from either function 17H or function 18H. Function 17H is valid for PC XT BIOS dated 01/10/86 and later and for all AT, PC-convertible, and PS/2 machines. Function 18H is valid for the AT BIOS dated 11/15/85 and later as well as for the PC XT Model 286 and PS/2 machines. If the machine does not support function 18H, it is not likely to support dual-capacity drives.

Here is the Turbo code that will find and move the data for us:

```
Const
  Disk360Params : array[0..10] of byte =
     ($DF,$02,$25,$02,$09,$2A,$FF,$50,$F6,$0F,$08);

Var
  RAMTable : DiskParameterTableType absolute $0000:$0078;
  regs     : registers;

Begin
  With Regs Do
    Begin
      AH := $18H;                  {set up to call function}
      DL := DriveNum;              {0 = A, 1 = B, etc.}
      CH := NumOfTracks;           {39 or 79 tracks}
      CL := NumOfSectors;          {number of sectors/track}
      Intr($13,Regs);              {call the BIOS}
      if Flags and 1 = 0 the move(mem[ES:DI],RAMTable^,11)
        else move(Disk360Params,RAMTable^,11);
    End;
End;
```

That's all there is to it. If the BIOS supports a drive type corresponding to the values in CH and CL, the carry bit will be clear upon return, and ES:DI will point to the correct parameter block in ROM. At the completion of a call to this procedure, the disk base parameter table in RAM will be loaded with the

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

desired values. Note that this particular example will load the RAM table with constant values for a 360K disk if service call $18 fails, which could occur if the BIOS doesn't support the call.

Now that the parameter data that supports the type of disk to be formatted is stored in the proper place, we can load up the track address field table in the following manner:

```
Procedure LoadParameterTable;
Var
  i : byte;
Begin
  For i := 1 to NumSectors do
    Begin
      AddressFields[i].HeadNum := 0;
      AddressFields[i].SectNum := i;
      AddressFields[i].SizeCode[i] := 2;
      AddressFields[i + NumSectors].HeadNum := 1;
      AddressFields[i + NumSectors].SectNum := i;
      AddressFields[i + NumSectors].SizeCode := 2;
    End;
End;
```

How many of you sharp-eyed programmers noticed that the first field of the record was not initialized? That is the track number field; it must be loaded during the program loop that sequentially calls the BIOS to format each track on the disk.

It is time now to consider the design of the heart of the formatting utility, the routine that actually calls function 05H of the disk-interface BIOS. For speed and compactness, I have incorporated it in my code as a Turbo in-line function. For illustrative purposes, I think it is best to present the code in high-level Pascal here:

```
Function FormatTrack(Var Params; drive, track,
                                 sectors) : word;
var Regs : Registers;
    i : byte;
Begin
  With Regs Do
    Begin
      ES := seg(params);
      BX := ofs(params);
      AH := $05;
      CH := track;
      DL := drive;
      DH := 0;
      AL := sectors;
      For i := 1 to sectors * 2 do Params[i].TrackNum :=
        track;
        Params[i].TrackNum := track;
      Intr($13,Regs);
      if Flags and 1 <> 0 then
        begin
          FormatTrack := word(AH);
          exit;
        end;
      DH := 1;
      ES := seg(params);
      BX := ofs(params) + (4 * sectors);
      AL := sectors
      AH := 05H;
      Intr($13,Regs);
      if Flags and 1 = 0 then FormatTrack := 0
        else FormatTrack := word(AH);
    End;
End;
```

There are a couple of things to note about this function. Observe how the track number field of the parameter table is loaded with the proper track number just before invoking interrupt 05H to format a track. Also note that this routine formats a full track on both sides of the disk. As in nearly all of the disk BIOS interface functions, an error is indicated by the setting of the carry flag. The only register changed by the call is the AX register. ES and BX must point to the beginning of the parameter table for each call to function 05H. The entire disk can be formatted by placing this function in a loop that continues for the desired number of tracks.

If no errors were encountered, the disk is now formatted. Are we done? Not by a long shot. The disk may be properly formatted, but until we initialize the boot sector, the file allocation table (FAT), and the root directory area, the disk is no good for anything but a frisbee.

Making the Disk Useful

Let's deal with the boot sector first. Every readable DOS disk has a boot sector that contains information vital to DOS. The area of the 512 bytes in the boot sector of primary interest to us is a fourteen-byte segment that is part of the BIOS parameter block (BPB). It is located at offset 13H in the boot sector and is structured as follows:

Offset	Type	360K Value	Description
0DH	byte	02H	Sectors per cluster
0EH	word	0001H	Reserved sectors
10H	byte	02H	Number of FATs
11H	word	0070H	Root directory entries
13H	word	02D0H	Total sectors
15H	byte	FDH	Media descriptor
16H	word	02H	Number of FAT sectors
18H	word	0009H	Sectors per track
1AH	word	0002H	Number of heads

None of these values must be changed to accurately describe the disk that we just formatted, but they would need to be changed if we were formatting other than a 360K disk. The best method for handling the boot sector is to use either DEBUG or the Norton Utility to write the boot sector from a floppy disk to a binary file and then import the file as a Turbo Procedure and assign a pointer to it, as follows:

```
{$L BOOTSEC.BIN}

Procedure BootSector; External;

Type
  BootType = Array[0..511] of byte;

Var
  Boot : ^BootType;

Begin

  Boot := @BootSector;
  ......
  ......
```

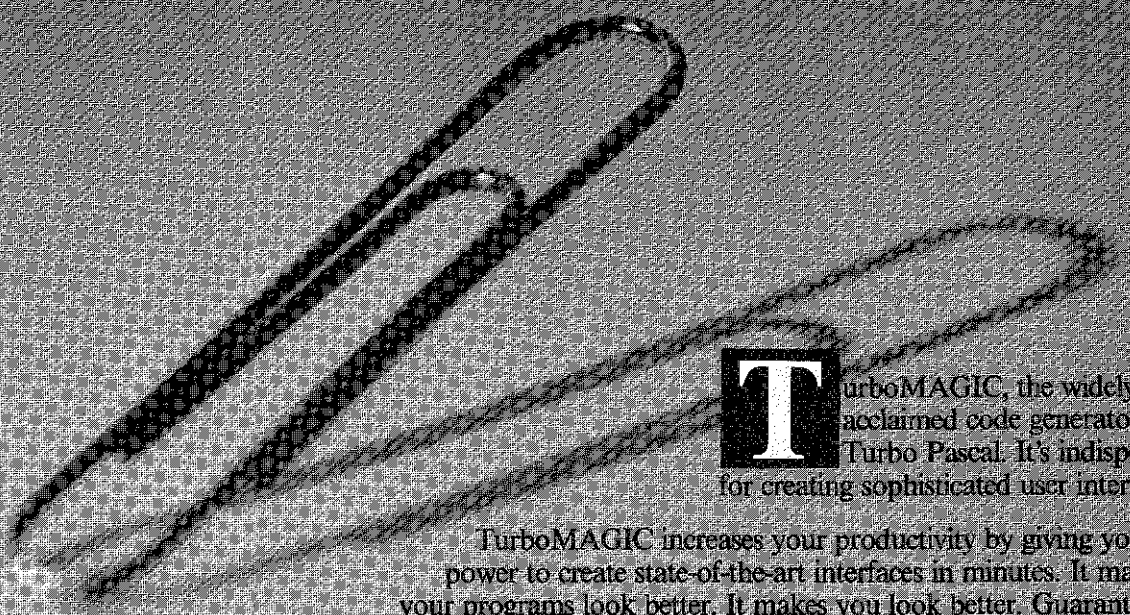
It doesn't matter what kind of disk you use for this method, because you can load the crucial fourteen-byte area with the correct data before writing the boot sector directly to disk.

Well, it almost doesn't matter. It seems that DOS 4.0 refuses to work with any disk that does not have the character string "IBM " starting at offset 03H in the boot record. But instead of belaboring that anomaly, let's define a typed constant to contain the data necessary to format a 5.5", 360K disk:

```
Const
  Disk360Boot : Array[1..14] of byte =

($02,$01,$00,$02,$70,$00,$D0,$02,$FD,$02,$00,$09,$00,$02);
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

The Right Tool.



TurboMAGIC, the widely acclaimed code generator for Turbo Pascal. It's indispensable for creating sophisticated user interfaces.

TurboMAGIC increases your productivity by giving you the power to create state-of-the-art interfaces in minutes. It makes your programs look better. It makes you look better. Guaranteed!

Use the highly-praised form painter to design even the most sophisticated input forms. And use the new interactive menu editor to create the menus you need to tie the pieces together. You can't beat it.

You need power. You need speed. You need flexibility. And above all you need a consistent, logical structure throughout your program. ... You need the **Right** tool — turboMAGIC.

Menu Features

- ✓ Borland-style menus
- ✓ Nested to 10 levels
- ✓ Multiple columns
- ✓ Paging/scrolling
- ✓ Disabled entries
- ✓ Pull-down systems
- ✓ Lotus-style menus
- ✓ Options menus
- ✓ Dynamic menus

General Features

- ✓ Pop-up windows
- ✓ Fast screens
- ✓ Mouse interface
- ✓ User color installation
- ✓ Monitor-based colors
- ✓ Directory search windows
- ✓ Context-sensitive & indexed help windows
- ✓ Definable keys
- ✓ Well-documented, easy-to-modify code

Input Form Features

- ✓ Up to 88 lines
- ✓ Scrolling in windows
- ✓ 26 user-modifiable field types
- ✓ Format masks
- ✓ Conditional-use fields
- ✓ Fast automatic updating of dependent fields
- ✓ Bomb-proof input handlers
- ✓ Output to printer or disk

"... the best screen generator on the market." George Kwasha, TUG Lines, Nov/Dec 87

"... the Cadillac of prototyping tools for Turbo Pascal." Jim Powell, Computer Language, Jun 87

ORDER your Magic TODAY! Only $199.
CALL TOLL FREE 800-225-3165 or 205-342-7026

turbo MAGIC
Version 2.0

Sophisticated Software Inc., 6586 Old Shell Road, Mobile, AL 36608

Requires 512K IBM PC compatible and Turbo Pascal 4.0 or 5.0. 30-Day Money Back Guarantee. S&H: US $5, Canada $15, Other $25

Number 257 on the Reader Service Card.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Now we need a routine to write sectors to the disk. I prefer working with the BIOS all the way with this project; so, here is a procedure that utilizes BIOS function 03H to write 512-block sector data to a floppy disk. Another reason for using this function is that it is safer. Bit 7 of the drive value must be set before the BIOS will attempt to write to a hard disk.

```
Function WriteSector(drive,track,side,
                     sector : byte; var buffer) : Boolean;
Var
  Regs : registers;
Begin
  With Regs do
    Begin
      AH := $03;
      AL := 1;
      ES := seg(buffer);
      BX := ofs(buffer);
      CH := track;
      CL := sector;
      DH := side;
      DL := drive;
      Intr($13,Regs);
      WriteSector := flags and 1 = 0;
    End;
End;
```

Now we're almost ready to write the boot sector to the target disk. First, though, we must place the fourteen crucial bytes into the BPB. Here's the statement that does just that:

```
Move(Disk720,Boot[13],14);
```

Next, write the boot sector to the disk (assume drive A):

```
If not WriteSector(0,0,0,1,Boot) then HandleError;
```

We know from the data in the BPB that the 360K disk format FAT occupies three sectors, that the media descriptor is $FD, and that there are two copies of the FAT on each DOS disk. An initialized FAT will contain all zero values except for the first three bytes, which will consist of the media descriptor followed by two bytes set to FFH. The FAT data structure for a 360K disk is a two-sector array indexed from 2 to 3 to correspond with the BIOS disk sector numbering scheme.

```
Var
  Fat : Array[2..3] of BootType;
  i : byte;
```

Initialize it with zero values, and then load the first three bytes:

```
Begin
  FillChar(Fat,sizeof(Fat),0);
  Fat[1] := $FD;  Fat[2] := $FF;  Fat[3] := $FF;
```

We are now prepared to write the FAT sectors to the target disk:

```
For i := 2 to 3 do
  Begin
    if not WriteSector(1,0,0,i,Fat) then;
    if not WriteSector(1,0,0,i+2,Fat) then;
  End;
```

This places two copies of the FAT, each two sectors in length, on the target disk.

Let's not lose track of how many sectors there are on side 0 of track 0 and how many sectors we have written. There are nine sectors per track on a 360K disk, and thus far we have written one boot sector and four FAT sectors. We still have seven sectors of root directory space to initialize; these will be split across side 0 and side 1 of track 1 of the disk. Here's the code that will initialize the disk directory area using the boot array defined earlier:

```
Begin
  FillChar(Boot,Sizeof(boot),0);
  For i := 6 to 9 do if WriteSector(1,0,0,i,boot) then;
  For i := 1 to 3 do if WriteSector(1,0,1,i,boot) then;
End;
```

Verifying the Disk

Okay, we have formatted the disk, initialized the FAT, and placed the directory sectors on the target disk. Are we finished? Well, if the target disk is fairly new and of high quality, one could store data on it with a high degree of confidence, but the format routine will not detect or report unreadable sectors. The problem is that the disk has not been verified. Remember that DOS conducts a verification check during format to detect and mark bad clusters in the FAT.

The first step in designing a disk-verification module is to develop the code that will physically verify the target disk by detecting and reporting problem sectors. It would also be sound strategy to design the verification code so that it will be at least as stringent as DOS in marking bad sectors. When DOS detects a bad sector during the format process, it assumes not only that the detected sector is bad but that the immediately adjacent sectors are also flawed. Therefore, DOS will mark the entire track (one side) as bad if it finds one bad sector on that track. Accordingly, the verification function will verify one track at a time and mark the entire track as bad if one sector on the track fails verification.

```
Function VerifyTrack(Drive,Side,Track,
      Sectors : byte) : boolean;
Var Regs : Registers;
Begin
  With Regs Do
    Begin
      AH := $04;
      AL := Sectors;
      CH := Track;
      CL := 1;
      DH := Side;
      DL := Drive;
      Int($13,Regs);
      VerifyTrack := Flags and 1 = 0;
    End;
End;
```

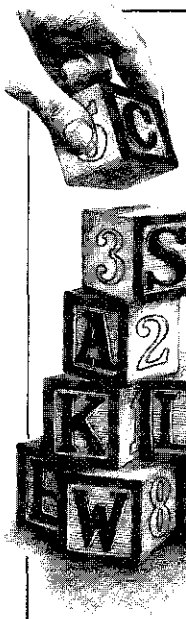
One note about this function is in order: Any PC ROM BIOS dated before 11/15/85 requires that ES:BX be set to point to a caller-provided buffer.

VerifyTrack can now be used to sequentially verify each track on each side of the target disk. Now we have to determine what to do when the function returns a FALSE value, indicating a bad sector on that track. The first task that needs to be accomplished is to convert from the three-dimensional BIOS coordinate system for sector identification (track, side, and sector number) to the DOS sequential numbering system. This is necessary to compute the offset into the FAT where the bad cluster marker will be placed. The algorithm for doing this is

```
DOS Sector = (BS - 1) + (Side * SPS) + (Track * SPS * 2);
```

where "BS" is the BIOS sector number, "SPS" is the number of BIOS sectors per side, "Side" is the side number (zero based), and "Track" is the BIOS track number (zero based).

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



NEW RELEASE!

Build with an Inside! advantage...

Thousands of C, BASIC, FORTRAN and Modula-2 programmers are already using Inside! to improve application performance. You have the best compiler and debugger, now add the performance analysis tool that helps create world class applications!

Inside! analyzes your program in real-time, timing each instance of a function, source line or arbitrary code fragment, all with microsecond accuracy and without source code modification. Utilize the following modes to know where to focus your optimization efforts for maximum reward.

- Function/procedure timing
- Call target timing
- Source Line timing
- Arbitrary event timing
- Non-executed functions or source lines
- MS-DOS function timing
- Overlay and assembly language support

Inside! is available for popular compilers . . .

Call or write today for a free brochure and the latest list of Inside! products.

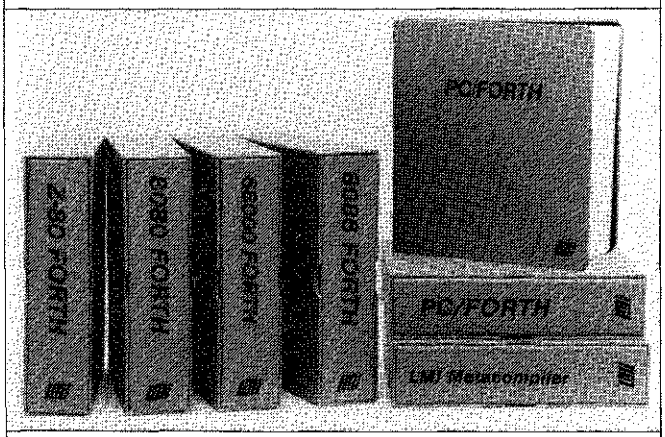
To Place Orders
(800)537-5043
Product Support
(508)478-0499
Visa/Mastercard/C.O.D. Accepted

$125.00 EACH
Satisfaction guaranteed

Paradigm Systems
P.O. Box 152 Milford, Massachusetts 01757
Inside! is a trademark of Paradigm Systems.

Number **195** on the Reader Service Card

TOTAL CONTROL with *LMI FORTH*™



For Programming Professionals:

an expanding family of compatible, high-performance, Forth-83 Standard compilers for microcomputers

For Development: Interactive Forth-83 Interpreter/Compilers

- 16-bit and 32-bit implementations
- Full screen editor and assembler
- Uses standard operating system files
- 400 page manual written in plain English
- Options include software floating point, arithmetic coprocessor support, symbolic debugger, native code compilers, and graphics support

For Applications: Forth-83 Metacompiler

- Unique table-driven multi-pass Forth compiler
- Compiles compact ROMable or disk-based applications
- Excellent error handling
- Produces headerless code, compiles from intermediate states, and performs conditional compilation
- Cross-compiles to 8080, Z-80, 8086, 68000, 6502, 8051, 8096, 1802, and 6303
- No license fee or royalty for compiled applications

For Speed: CForth Application Compiler

- Translates "high-level" Forth into in-line, optimized machine code
- Can generate ROMable code

Support Services for registered users:

- Technical Assistance Hotline
- Periodic newsletters and low-cost updates
- Bulletin Board System

Call or write for detailed product information and prices. Consulting and Educational Services available by special arrangement.

Laboratory Microsystems Incorporated
Post Office Box 10430, Marina del Rey, CA 90295
Phone credit card orders to: ***(213) 306-7412***

Overseas Distributors.
Germany: Forth-Systeme Angelika Flesch, Titisee-Neustadt, 7651-1665
UK: System Science Ltd., London, 01-248 0962
France: Micro-Sigma S.A.R.L., Paris, (1) 42.65.95.16
Japan: Southern Pacific Ltd., Yokohama, 045-314-9514
Australia: Wave-onic Associates, Wilson, W.A., (09) 451-2946

Number **63** on the Reader Service Card

Desktop 9-Track Tape Subsystem

Now, 9-track tape lets your micro exchange data with minis and mainframes.

Simply exchange data files on a reel of 9-track tape.

9-TRACK is the first choice for file interchange among data processing professionals. Now, Qualstar's low cost 1/2-inch 9-track *Ministreamer* tape systems bring full ANSI data interchange to IBM PCs or Macintosh, giving your micro the freedom to exchange data files with nearly any mainframe or minicomputer in the world.

Available in both 7" and 10-1/2" versions, compact Qualstar tape drives can sit on your desktop, *using less space than an ordinary sheet of paper*. Systems include DOS or XENIX compatible software, coupler card and cables. High reliability 1600 or 6250 BPI capability may be used for disk backup as well as data interchange. Discover the big advantage 9-track tape has over other micro/mainframe links.

DEC HP ALTOS NCR — IBM MAINFRAMES — AT&T UNISYS APOLLO SUN

Call us today!

FOR DETAILS AND TO ORDER:
FAX (818) 882-4081
PHONE (818) 882-5822

#1 Selling 9-Track Systems on the Desktop

QUALSTAR®

9621 Irondale Ave., Chatsworth, CA 91311

©1988 Qualstar Corp. All product and company names and trademarks are the exclusive property of their respective owners.

Number **97** on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

The value of 2 represents sides per disk and is a constant here, in the assumption that single-sided disks are obsolete. The function **DosSector** implements this algorithm:

```
Function DosSector(Side,Track,BiosSector,SPS : byte)
                : word;
Begin
  DosSector := word(BiosSector - 1) + word(Side * SPS) +
               word(Track * SPS * 2);
End;
```

Next, we need to convert the DOS logical-sector number to a cluster number. Both 1.44M and 1.2M disks have a cluster size of one sector, while all others have two sectors per cluster. Here we will use a global variable named SPC (sectors per cluster), which should be set at the beginning of the program according to the type of disk format. A second global variable, *CNAdjust*, must also be computed prior to formatting. DOS clusters are numbered from 2 to one greater than the total number of data clusters on the target disk. This allows for the boot sector, FAT sectors, and directory sectors not being included as data clusters.

In the case of a 360K floppy disk, the first DOS logical sector that contains data is number 12. Dividing by 2 produces a cluster number of 6, but since cluster numbering (for FAT offset computation purposes) must start at 2, an adjustment must be made. *CNAdjust* is computed by dividing the first DOS data sector by the number of sectors per cluster and then subtracting 2. *CNAdjust* will equal 4 for a 360K disk and 5 for a 720K disk.

```
Function DosCluster(DosSector : word) : word;
Begin
  DosCluster := (DosSector div SPC) - CNAdjust;
End;
```

Now that we have the DOS cluster number, the next step is to calculate the corresponding offset into the FAT. This is accomplished by the function **FATOffset.**

```
Function FATOffset(Cluster : word) : word;
Begin
  FATOffset := (Cluster * 3) div 2;
End;
```

Now comes the tricky part. All floppy disks use a 12-bit FAT entry, and each FAT entry pair occupies 1.5 bytes. Bad disk clusters are indicated in the FAT by a value of $FF7. The manner in which this value is stored depends upon whether the cluster number is even or odd. The procedure **MarkFAT** will place the appropriate entry into the copy of FAT our program will keep in memory and ultimately will write to the target disk. Note the global declaration of the FAT array that precedes the procedure definition. This array is sized at 4608 bytes to accommodate the largest FAT (9 sectors for a 1.44M drive) that will be encountered. Also, be sure to initialize the array with zero values somewhere early in the program. *BadCount* is a global variable that will indicate if any bad clusters are found and marked during the verification process.

```
Type
  FATType = Array[1..512] of byte;

Var
  FAT : array[0..4607] of byte;
  FATSector : array[1..9] of FATType absolute FAT;
  BadCount : word;

Procedure MarkFat(ClusterNumber : word);
Var Offset : word;
Begin
  Inc(BadCount);
  Offset := FATOffset(ClusterNumber);
  if odd(ClusterNumber) then
    Begin
      Fat[Offset] := Fat[Offset] or $70;
      Fat[Offset+1] := $FF;
    End
  Else
    Begin
      Fat[Offset] := $F7;
      Fat[Offset+1] := $0F;
    End;
End;
```

Here is the main procedure for verifying the freshly formatted disk:

```
Procedure VerifyDisk(Drive,Tracks,Sectors,SPS,FatLen
                : byte);
Var Trk,Sector,Side : byte;
    LastCN : word;

Procedure MarkTrack(Drive,Track,Side : byte);
Var Sector : byte
    DosSect,Cluster : word;
Begin
  For Sector := 1 to Sectors do
    Begin
      DosSec := DosSector(Side,Track,Sector,SPS);
      Cluster := DosCluster(DosSec);
      if Cluster <> LastCN then
        Begin
          MarkFat(Cluster);
          LastCN := Cluster;
        End;
    End;
End;

Begin
  FillChar(Fat,Sizeof(Fat),0);
  BadCount := 0;
  LastCN := 0;
  FAT[0] := $FD;
  FAT[1] := $FF;
  FAT[2] := $FF;
  For Trk := 1 to Tracks do
    Begin
      For Side := 0 to 1 do
        if not VerifyTrack(Drive,Trk,Side,SPS) then
          MarkFat(Drive,Trk,Side);
    End;
  If BadCount > 0 then
    Begin
      For i := 2 to FatLen + 1 do
        Begin
          If WriteSector(Drive,0,0,i,FATSector[i]) then;
          If WriteSector(Drive,0,0,i + FatLen,
                FATSector[i]) then;
        End;
    End;
End;
```

This procedure will verify each track on the target disk. If any bad clusters are found on a given track, the entire track is marked as bad, and the FAT is marked accordingly. Note the use of the variable *LastCN* to prevent wasting time by marking the same cluster twice.

Taking It Further

The code that I have presented here is not complete. It is presented primarily as an example of the kind of programming strategy that is necessary to accomplish a floppy disk

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

format. A more complete set of code should include several data structures that define the disk parameters for each kind of disk format to be encountered. For example, the first byte of the FAT is initialized in the procedure **VerifyDisk** as $FD, which is the media descriptor byte for a 360K disk. Also, the formatting strategy embodied in this code assumes that the host machine BIOS supports disk interface service $18. Older machines and some clones may not support this call. There are also some disk-interface BIOSs in which the service is supported through a service number different from that specified in the IBM *Technical Reference Manual*. The Zenith 386 disk-interface BIOS lists the IBM services of 17H and 18H as 11H and 12H, respectively. A more general strategy might also make use of disk-interface BIOS service $15, which can be used to determine if the target drive is a dual-capacity drive.

Error handling has been deliberately omitted from the example code, as the handling of errors is very application specific. Suffice it to say that a well-designed program will incorporate a sophisticated error-handling strategy and attendant error message delivery to the user.

John J. Newlin is a retired naval aviator who holds degrees in mathematics and computer science. He has programmed extensively in assembly, Turbo Pascal, and Modula-2. He is the author of four Shareware products and is a member of the board of directors of the Association of Shareware Professionals. Readers may write to John at 6198 Agee St. #71, San Diego, CA 92122. ♦

New! **Turbo Pascal Tools for Powerful Multi-User Databases**

Write powerful multi-user databases faster and easier using the language you love - Turbo Pascal - and **B-Tree Filer 5.0**. Easily upgrade your present applications that use Borland's Database Toolbox.

The Performance and Connections You Need

You'll have the fastest, safest, most flexible databases when you use **B-Tree Filer 5.0** – no rigid structure, no TSR hassles, no running out of files. And they're compatible with Novell, 3Com, MS-NET, and others.

The Complete B-tree Tool Kit

In over 100 Turbo 5.0 procedures you get:
• Fixed and variable length records • Two billion records per database • 100 keys per index file • Flexible record locking • B-tree indexing with advanced page balancing • Fail-safe mode with journaling • Units for sorting, browsing, reindexing, and reorganizing • Same source runs on any network • Complete source code. B-Tree Filer includes full documentation and technical support and you pay NO royalties.

B-Tree Filer 5.0 is only $125. (single user)
Multi-user version with network support, only $175.

Call toll-free for credit card orders.
1-800-538-8157 ext 830 (1-800-672-3470 ext 830 in CA)

Complete capabilities brochure available, call 408-438-8608 (9 to 5 PST) or write for more information. Also from TurboPower:
• Turbo Analyst 5.0, tune and document your programs, only $99.
• Turbo Professional 5.0, the best general purpose library, $125.

Satisfaction guaranteed or your money back within 30 days. Turbo Pascal 4.0 or 5.0 is required. Shipping and taxes prepaid in U.S. and Canada. Elsewhere add $15 per unit.

TURBO Power
TurboPower Software
P.O. Box 66747
Scotts Valley, CA 95066-0747

Number **25** on the Reader Service Card

New & Improved Version 2.1

TRANSLATOR
(Pascal to C)

- For Programmers and those learning C
- Obtain the Attributes and Extensions of the Powerful C Language from Your Natural Language Pascal Programs
- Port your Pascal Programs to UNIX and other C Environments
- Now supports Strings and Pascal I/O
- Convert Entire Programs or Program Segments Interactively.
- User Manual and User Disk Contain Many Illustrative Example TRANSLATIONS with Explanations of Significant Difference between the Pascal and the C Programs.

New Low Price Only: $99.95

MILTON BROWN SOFTWARE
935 Chester Avenue
San Marino, CA 91108
(818) 796-6934

Number **240** on the Reader Service Card

CHANGE OF ADDRESS?

Please send us your new address along with your previous ZIP code, using this handy form— Thanks.

NAME _____
COMPANY _____
ADDRESS _____
CITY _____ STATE _____
ZIP CODE _____ OLD ZIP CODE— _____
TELEPHONE _____

Please mail to:
Programmer's Journal
P. O. Box 30160
Eugene, OR 97403

Or call toll free
800-234-0386

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Windowing and Graphics

XVT Version 1.2
Advanced Programming Institute Ltd.
Box 17665
Boulder, CO 80308
(303) 443-4223
$995

Currently, developing windowing and graphics-oriented applications targeted for several environments is truly a challenge. Microsoft Windows is a world apart from the Apple Macintosh environment. OS/2 Presentation Manager differs enough from Windows to break most existing Windows code. And X Window and NeWS on Unix and other workstation platforms pose a completely different development environment. The Extensible Virtual Toolkit (XVT) embodies a powerful concept for these differing environments: Supply a standard higher-level interface that hides the underlying windowing environment from the programmer.

Version 1.2 of XVT is a collection of C function libraries that allow the writing of programs that are portable across various user interface toolkits; those currently supported are the Apple Macintosh and Microsoft Windows. XVT allows programs to be written that are source code compatible from Windows to the Macintosh. Compiled and linked under the appropriate environment, the resulting application is designed to conform with the standard user conventions for each environment. XVT completely hides these native toolkit functions and differences from the programmer and supplies its own functions for user interface features that make calls to the underlying toolkit.

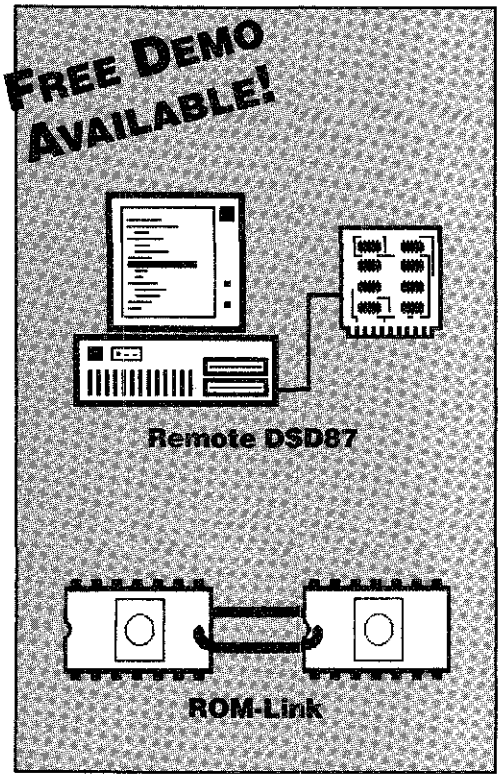
The current distribution includes header files, the toolkit library, utilities, sample source code, and examples for both Windows and the Macintosh (supplied on both 5.25" PC and Mac disks). In addition to XVT, under DOS you will need Windows 2.0X, Windows/286, or Windows/386 run time; Windows Software Development Kit (2.03 or greater); and Microsoft C v5.10. The medium model under Microsoft C is supported (far data is supported from the heap). For the Macintosh, you will need System 6.02 or later plus one of the following compilers: Lightspeed C v3.0 or later and RMaker; or MPW 2.0.2 and MPW C 2.0.2. A Pascal interface is available on special order for MPW Pascal (Mac) and Microsoft Pascal 4.0 (PC).

XVT does not support all features of the underlying toolkits. Only those features of the Mac and Windows user interface that can be virtualized can realistically be supported. For example, Windows has Dynamic Data Exchange (DDE) for intertask communication between applications. The Mac doesn't support intertask communications even under Multifinder; however, most of the features—events, windows, fonts, graphics, dialog boxes, menus, resources, files, and so on—are supported. The XVT-supplied functions are generally at a higher level of abstraction from the native toolkits. For Windows developers, XVT functions also attempt to provide a consistency sorely lacking in the native Windows SDK.

If you aren't targeting both environments, there are some weaknesses to XVT. Because XVT interposes another layer between your application and the underlying environment, it has to slow execution and make the application larger. XVT can't take advantage of native hardware or functions not supported on both environments, such as DDE, regions, off-screen bitmaps, and fast bitblt operations. The PC and Mac keyboards don't overlap. XVT has tried to extend the virtual keyboard to cover most dedicated PC keys, but extended PC keystrokes using Alt, Ctrl-Function, and Shift-Function commonly used

Continued on page 89

For the Ultimate in Embedded Systems Programming.



Debug Debug Debug Debug Debug

Remote DSD87 is the best source debugger for embedded development and the only product offering high performance and advanced software In Circuit Emulator capabilities. This package and your compiler form a complete solution for the development of embedded programs. Order now and receive a FREE copy of Native DSD87, a $125 value.

Remote DSD87 (supports both Microsoft and Turbo C) . . . $395
Remote DSD87 OMF (includes Intel OMF support) $795

Link Link Link Link Link Link Link Link

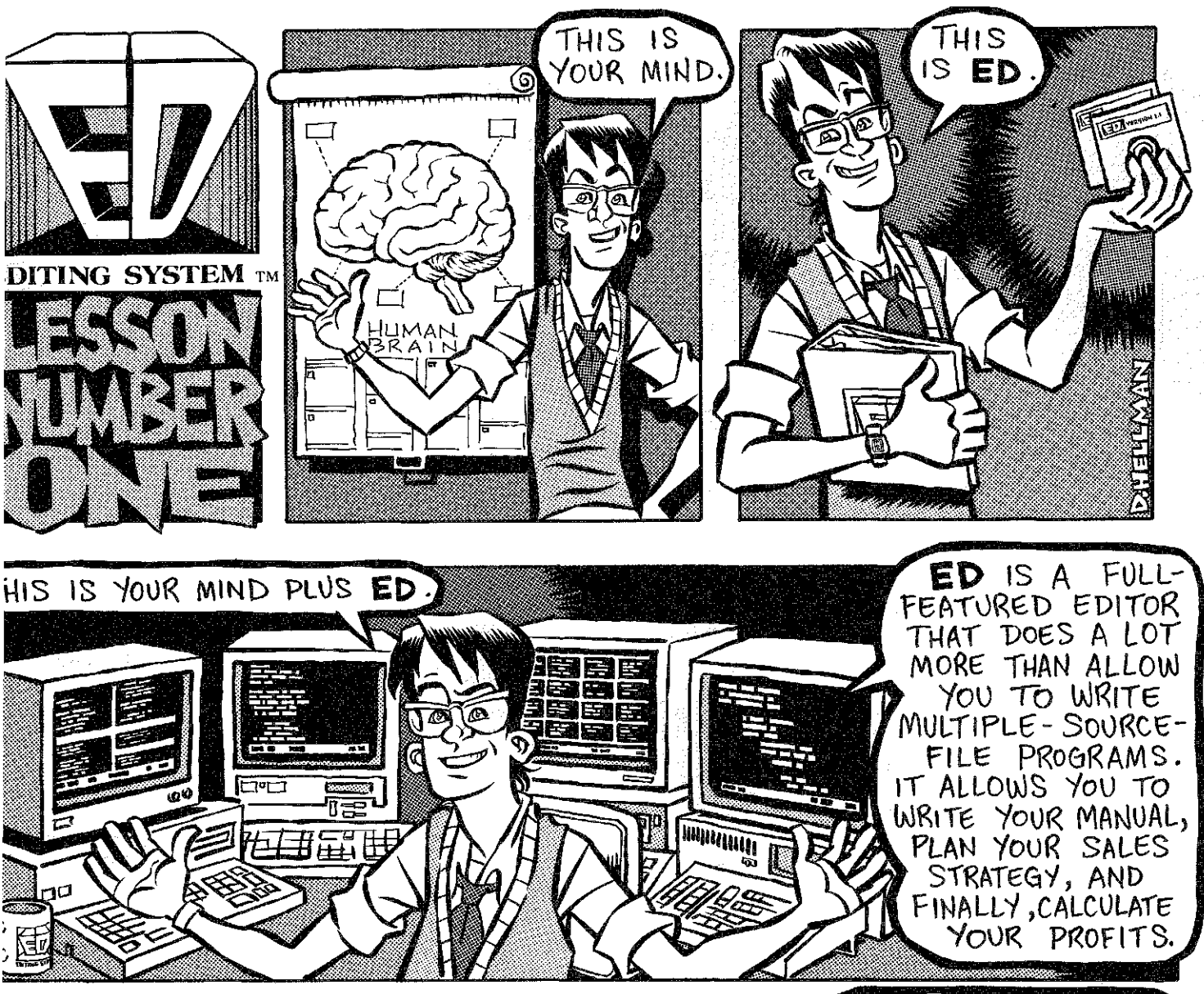
ROM-Link is the first integrated link and locate utility to directly generate COM, EXE, SYS, HEX, or Intel OMF files. The only linker ever guaranteed faster than any present or future competitors. Supports Microsoft debugging information. Receive a FREE librarian and cross reference utility with your order!

ROM-Link $395

Soft Advances
10811 Washington Bl., Ste. 205 • Culver City, CA 90232 • (213) 559-7015

Number 33 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

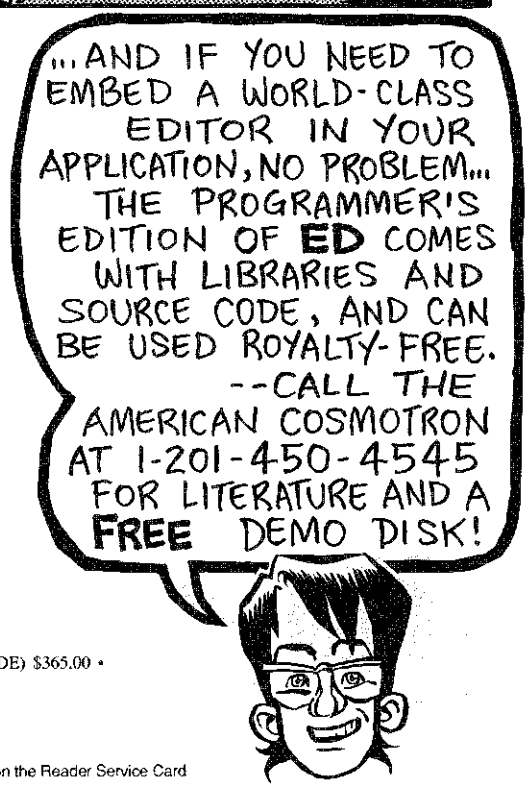

ED
DITING SYSTEM ™
ESSON
UMBER
ONE
THIS IS YOUR MIND.
HUMAN BRAIN
THIS IS ED.
ED version 1.1
DIELLMAN
HIS IS YOUR MIND PLUS ED.
ED IS A FULL-FEATURED EDITOR THAT DOES A LOT MORE THAN ALLOW YOU TO WRITE MULTIPLE-SOURCE-FILE PROGRAMS. IT ALLOWS YOU TO WRITE YOUR MANUAL, PLAN YOUR SALES STRATEGY, AND FINALLY, CALCULATE YOUR PROFITS.

nter the world of ED and experience limitless editing power and amazing speed.

) is an object-oriented open architecture system for editing programs, manuscripts, and data es; and creating sophisticated sorts and filters, customized applications, and snazzy demos.

ith ED's Libraries and Source Code, a programmer can manipulate objects such as editors, ndows, data entry windows, macros, menus, browsers, popup directories, dynamic arrays, gular expressions, and finite state automata through normal C function calls.

• EDIT UNLIMITED NUMBER OF FILES IN MULTIPLE WINDOWS • **POPUP DIRECTORY FACILITY** • VIEW ESPEC/SUBDIRECTORIES • FIND FILE ON DISK • SEARCH DISK FILES FOR TEXT • BROWSE • FILE STATUS • EDIT XECUTE • SORT • **SEARCH AND REPLACE** • FORWARD/BACKWARD • CASE SENSITIVITY • REGULAR EXPRES-ONS • WITHIN BLOCK • ACROSS ALL BUFFERS • **BLOCKS** • COLUMN/LINE/STREAM BLOCKS • SAVE • CUT • LETE • BLANK • SEARCH-FOR • FORMAT • JUSTIFY LEFT/RIGHT/CENTER/BOTH • UPPER/LOWER CASE • REMOVE VERLAY • REACTIVATE • TAB • DRAG • SORT • EXECUTE • **SPREADSHEET STYLE MATH** • D/SUB/MULT/DIVD/AVG • **MACROS** • MENU DRIVEN • AUTO-EXECUTE • TIME-DELAY • NESTED • RECURSIVE NTERACTIVE• MULTIPLE KEYSTROKE • SCREEN DISPLAY CONTROL • AUTOMATIC MENU GENERATION • ORE FUNCTIONS • PULLDOWN USER MENU SYSTEM • SELECTIVE UNDO • RECONFIGURE COLOR AND WIN-W STYLE • ETCHA-SKETCH STYLE DIAGRAM/BOX DRAW • TEMPLATE EDITING • COMPILE WITHIN ED • ME/DATE STAMP • BRACKET MATCHING • GRAPHICS CHARACTERS • DOS SHELL • POPUP ASCII TABLE • USER FINED POPUP FILES • KEYBOARD CONFIGURATION • PRINT BUFFER WITH PAGINATION • ON-LINE MANUAL, TORIAL, AND HELP • **SUPPORTS** • DOS • PC/XT/AT, PS/2, 386 • CGA, MDA, EGA, HERCULES, VGA, MDS GENIUS, (SE 700/AMDEK 1280 • RUNS IN ALL VIDEO MODES, NO FLAGS OR DRIVERS • 256K • TURBO AND MICROSOFT C •


...AND IF YOU NEED TO EMBED A WORLD-CLASS EDITOR IN YOUR APPLICATION, NO PROBLEM... THE PROGRAMMER'S EDITION OF ED COMES WITH LIBRARIES AND SOURCE CODE, AND CAN BE USED ROYALTY-FREE. --CALL THE AMERICAN COSMOTRON AT 1-201-450-4545 FOR LITERATURE AND A FREE DEMO DISK!

Version 1.1 • $265.00 • THE PROGRAMMER'S EDITION (W/LIBRARIES AND SOURCE CODE) $365.00 •

MASTER CARD, VISA, C.O.D. and P.O.'s

IE AMERICAN
OSMOTRON 80 HOLMES ST • PO BOX 128 • BELLEVILLE, NJ 07109

Number 215 on the Reader Service Card

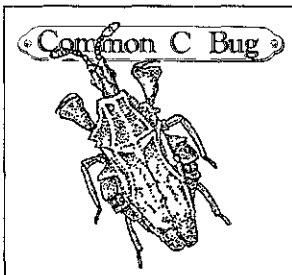
PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

T-SHIRTS

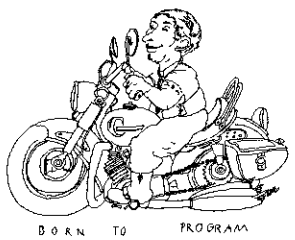
Multi-color imprints on the highest quality 50/50 cotton T-shirts.

$12.00 each – includes shipping & handling

SIZES S, M, L, XL



COMMON C BUG



BORN TO PROGRAM

PARALLEL PORTS

RS-232 HEAD

BUGS RULE 4 color imprint on a white shirt

PJ ISSUE SPECIAL!

All 6 1988 Back Issues— Plus VGA Disk **$35.00**

Save 33% off the regular price

The VGA *Graphics Express*

All 6 issues from 1988— plus the VGA CodeDisk

This is it! The highly acclaimed **Michael Abrash VGA series plus the VGA disk.**

Now only **$35.00**, a savings of **$17.00** off the regular price!

Offer expires April 30, 1989

BACK ISSUES

$7.00 each PREPAID ONLY

A few of the Back Issues currently available

Vol. 5.2: DTACK GROUNDED; Starting your own software house; Pink noise; TSR graphics print screen utility; Coding guidelines for resident programs; Parallel processing with the EGA; C—portability in practice and using header files; Proposed ANSI C Language Standard, Part 6; Indexing in BASIC; A first look at Turbo C.

Vol. 5.3: Starting your own software house--publicity; Unix books, Part 2; Consulting—merger mania; EGA data control; Modula 2 for Pascal lovers; Optimizing Turbo Pascal; ANSI C language standard; Coping with ANSI C and forcing file updates; Formatting BASIC output with buffered PRINT USING.

Vol. 5.4: Starting your own software house—technical support; Unix books, Part 3; Consultant productivity and effectiveness; EGA set/reset capabilities; ANSI C language standard; Character tables and string functions in C; MWWIND—a menu with windows utility; Variable passing and hiding in QuickBASIC 2 subprograms.

Vol. 5.5: Intel's 80386 vs. OS/2; Metagraphics President Jack Davis interview; Consulting—making computer tools pay for themselves; ASM—the invisible CapsLock; TSRs using expanded memory with Turbo Pascal; Note on optimizing Turbo Pascal; Proposed ANSI C Language Standard; Printer output from Turbo C programs; Matchup of Turbo Basic 1.0 and QuickBASIC 3.0.

Vol. 5.6: Multics, OS/2, paging and segmentation; The Usenix conference and other news; How to get rich in consulting; Shell in Marshal Pascal; Turbo Pascal 4.0 preview; Handling fixed length charaacter arrays in C; The proposed ANSI C Language Standard.

Vol. 6.1: Consulting is an attitude; Write mode 3 of the VGA; The LIM 4.0 expanded memory specification; Environment variables in Turbo Pascal 4.0; A timer program in C; Porting mainframe software to the PC--FORTRAN compilers under DOS and the Definicon 32-bit coprocessing engines.

Vol. 6.2: Hal Hardenbergh on RISCs, CISCs, and Fabs; Pink noise; Yet another VGA write mode; Inside Turbo Pascal 4.0; The proposed ANSI C Language Standard; Converting a Turbo C program to a TSR; 80286 and 80386 programming; 80386 software development; The Norton Guide for BIOS/DOS Assembly.

Vol. 6.3: Hal Hardenbergh on memory trends; Pink noise; Getting your consulting tools in order; Gordon Letwin on OS/2 memory management; Virtualizing the 386; Pascal—A CASE of the jumps; Turbo Professional 4.0; BASIC—CALL, GOSUB, and multi-line functions

Vol. 6.4: Job-shopping; Windowing systems overview; Reading VGA memory; X Window—a view from DOS; A digital clock in C for MS Windows; Windowing in TopView, DESQview, & MS Windows; MS C 5.1 and the ANSI Standard; A mouse unit for Turbo Pascal 4.0.

Vol. 6.5: Hal Hardenbergh on computers in the fast lane; Consulting—no-win jobs and too much work; VGA text paging; Using dynamic arrays and records in Turbo Pascal 4.0; An OS/2 Presentation Manager application template; Watcom C 6.0 and the ANSI Standard; Object oriented OS/2 in C++; Ada and software engineering; REXX on the PC.

Vol. 6.6: PC DOS 4.0; OS/2 vs. Unix; VGA screen saving and color selection; MASM quirks; Addressing extra memory for DOS; DOS device drivers debugging and information utility; Proposed ANSI C Language Standard; Ray Duncan on Forth; Ada data types and exceptions; PC floating point hardware and software.

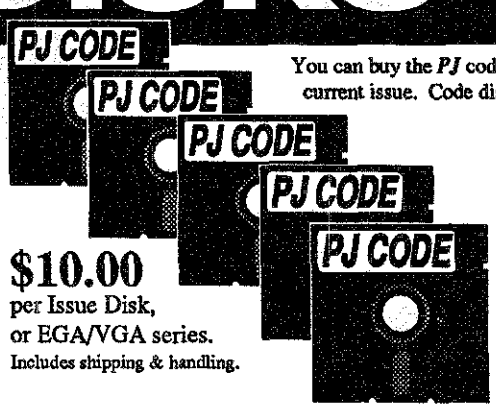
Vol. 7.1: GRAPHICS: High-resolution VGA color modes; Graphics processors; Modifying type constants stored in Turbo Pascal 4.0 and 5.0 compiled EXE code; Comparing Pascal overlay managers; JPI's TopSpeed Modula-2; faster QuickBASIC code; Microsoft's CodeView debugger; Lattice C V3.3; Smart terminal emulations in Windows; the COBOL-85 ANSI standard; Pink Noise industry news and reviews.

10% discount on any order of 10 or more T-Shirts, Disks or Back Issues.

All mail orders must be accompanied by check or Money Order -- PREPAID ONLY.

CODE DISKS

Use the order form on the card opposite. Order by phone with your VISA/MC. CALL TOLL FREE 800/234-0386 ext 53



$10.00 per Issue Disk, or EGA/VGA series. Includes shipping & handling.

You can buy the *PJ* code listings from volume 5.5 to the current issue. Code disks for the Michael Abrash EGA/VGA series are also available.

Diskettes are available for the following issues:

5.5	6.4	EGA
5.6	6.5	VGA
6.1	6.6	
6.2	7.1	
6.3	7.2	

360K MS-DOS disks standard, other formats available

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

under DOS and some Windows applications have no equivalent on the Mac. Currently, color is not supported, although grays and patterns are. Version 1.2 has provided some convenient hooks to allow use of the native toolkit functions and keyboards in these cases—but at the loss of your application's portability.

Documentation consists of a rather full three-ring binder with sections on installation and usage, a programmer's guide (organized by subject), a programmer's reference (alphabetical by macro and function), technical notes, release notes (a list of bugs and fixes), and a quick reference. A convenient XVT reference card and a 60-page technical overview of the toolkit are also supplied. The technical overview is available free for those desiring further information on XVT. The documentation is thorough but lacks an index and a comprehensive table of contents.

XVT comes with free updates, a warranty against defects, and technical support for a year. There are no royalties on distributed object code. I have worked with XVT for only a short time (and almost exclusively the Windows version), but I am impressed with the much simpler (in comparison with Windows), more powerful, and consistent XVT program interface.

TIGA-340 and 34010 Tools
Texas Instruments Inc.
P.O. Box 1443
Houston, TX 77001
(713) 274-2340—Graphics Hotline

Since the introduction of the IBM PC, graphics hardware on PC machines has been dominated by standards developed by IBM. And although IBM clearly lost control of the EGA/VGA hardware market, the IBM chip designs are being emulated by the other hardware vendors. At the high end of the PC market are a variety of divergent graphics boards. An ever-growing number are based on TI's 34010 graphics coprocessor. The TMS34010 is a full-fledged 32-bit microprocessor designed specifically for graphics operations (see "Migrating to Graphics Processors" in *PJ* 7.1). TI calls this chip a graphics system processor (GSP).

Although IBM has its 8514/A chip at the high end, and Intel has the 82786, the 34010 is clearly a more powerful option. TI has already announced the second generation of graphic processors in this family—the 34020 (GSP) and 34082 (floating-point processor). The 34020 is projected to offer performance up to 50 times faster than existing 34010 boards. The TMS340 GSP design is the direction that we'll see for most new I/O processor boards on the PC. The 34010 can execute graphics operations concurrently, freeing the 80X86 CPU for other tasks. Compaq has recently adopted the 34010 for it's high-end graphics board. Microsoft is writing a TIGA-340 driver for OS/2 Presentation Manager. The TMS340 family just might become the next high-resolution graphics standard.

The TIGA-340 specification defines a software interface for the TMS340 family of

Commenting Disassembler!

SOURCER

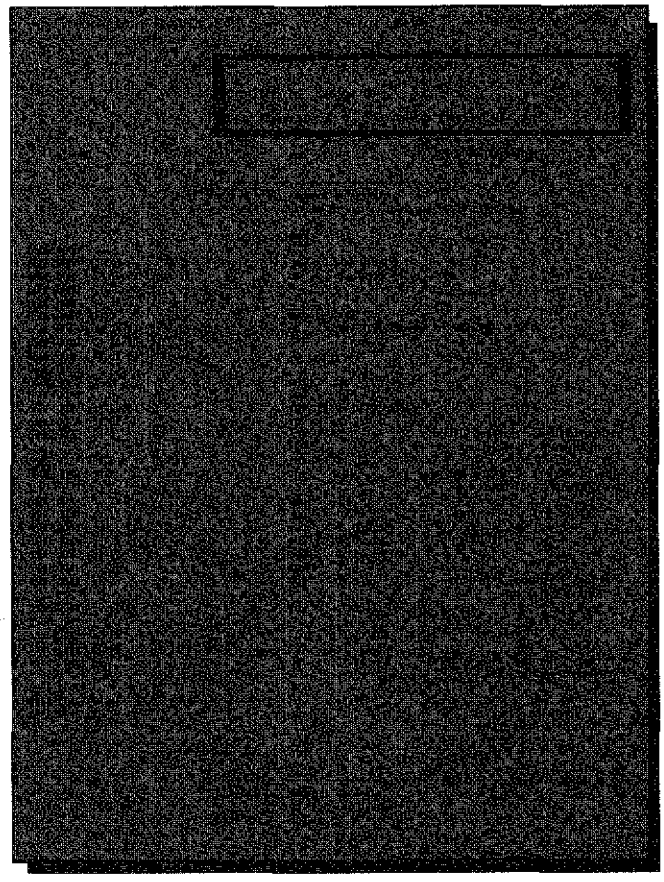
■ SEE HOW PROGRAMS WORK
■ EASILY MODIFY PROGRAMS

SOURCER™ creates detailed commented source code and listings from memory and executable files. Built in data analyzer and simulator resolves data across multiple segments and provides detailed comments on interrupts and subfunctions, I/O ports and much more. Determines necessary assembler directives for reassembly. Includes a definition file facility to include your own remarks and descriptive labels, force data types, and more. Complete support for 8088/87 through 80286/287 and V20/V30 instruction sets. We welcome comparisons with any other product, because no product comes close to the ease of use and output clarity of SOURCER.

On my list of programs that I simply won't do without!
—Robert Hummel, Senior Technical Editor, PC Magazine, 4/26/88

SAMPLE OUTPUT

Fully automatic
Program header
Assembler directives
Determines data areas and type
Detailed comments
Simulator follows segment changes
Easy to read format



(Source code output and inline cross reference can also be selected)

BIOSOURCE

■ CHANGE AND ADD FEATURES
■ CLARIFY INTERFACES

for PS/2, AT, XT, PC, and Clones

The BIOS Pre-Processor™ with SOURCER provides the first means to obtain accurate legal source listings for any BIOS! Identifies entry points with full explanations. Resolves PS/2's multiple jumps for improved clarity. Provides highly descriptive labels such as "video_mode" and much more. Fully automatic.

SOURCER Commenting disassembler	$99.95	**UNPACKER**™ Unpack packed EXE files and more	$39.95
SOURCER with BIOS Pre-Processor	139.95	**ASMtool**™ Assembly source analyzer and flowcharter	89.95

USA Shipping & Handling $3; Outside USA $15; CA Res. add sales tax; PS/2, AT, XT, and PC are trademarks of IBM Corp.

All our products come with a 30 day money back satisfaction guarantee. Not copy protected. To order or receive additional information just call!

VISA 1-800-662-8266 MasterCard

V COMMUNICATIONS, INC.
3031 Tisch Way, Suite 905, Dept. J3, San Jose, CA 95128 (408) 296-4224

Number **125** on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

graphics system processors. TIGA-340 is designed to standardize how 80X86-based microprocessors talk with a TI 340X0 graphics processor. TIGA provides a standard interface and optimizes communication between the 340 processor and the host CPU. Thus, programs that write to the TIGA interface should eventually be compatible with all 340 graphics boards (each vendor would supply their own TIGA-340 driver). The TIGA interface consists of three components: the application interface (on the host), the communications driver (TSR on the host), and the graphics manager (on the 340X0 board).

The applications interface is currently written to accept Microsoft C calling conventions and appears to the developer as C header files and another C library. This interface makes calls to the communications driver (actually a TSR program), which is supplied by the 34010 board vendor. This TSR passes the calls over to the Graphics Manager running on the 340X0 board. The Graphics Manager on the 340X0 does most of the actual work.

The application interface consists of a base set of TIGA primitives (79 functions) and extended primitives (44 functions). The base set would be available on every 340 board. The extended primitives are designed as optional primitives (leaving program memory available on the 340X0 board) and are downloaded on demand. The extended primitives can be replaced by downloaded 340X0 code modules and custom routines by the developer. The TIGA primitives cover initialization (15), attribute control (19), utility functions (8), text (6), graphics output (34), pixel arrays (5), cursor control (5), memory management (6), communications (16), and extensibility (9). All of the graphics output functions are extended primitives and may be replaced.

TI is offering a TIGA-340 Driver Development Kit ($340) with a user's guide, a TIGA-340 interface for the TI Software Development Board (SDB), utilities, and several TIGA-compatible application driver examples (AutoCAD release 9). With these tools, a programmer can design an application to run with the TIGA-340 interface. TI also offers a bundled Software Developer's Kit ($1495) that includes the TIGA Driver Kit along with the 340 assembler package, the 340 C compiler package, 340 math/graphics and fonts libraries, PC utilities, and documentation. The assembler, C compiler, linker, and other tools run on an 80X86 machine as cross-compilers. These development tools can also be purchased separately. The cost of their SDB has also been reduced to $1495, although other 34010 boards are suitable for development.

TI has released several new and revised manuals for the 340 family. For general familiarity, I would recommend the revised *TMS34010 User's Guide* ($24.95), which solidly details software and hardware aspects of the 34010 in about 600 pages. More general information on the TMS 340 family and TIGA-340 is available free from TI or its regional distributors.

Disassemblers

SOURCER v1.88 and Utilities
V Communications
3031 Tisch Way, Suite 905
San Jose, CA 95128
(800) 662-8266

V Communications has introduced a new version of SOURCER ($99.95), their commenting disassembler. SOURCER automatically converts executable files and programs in memory into commented source code and listings. Disassemblers come in two varieties: interactive and batch. SOURCER is of the batch or automatic type. SOURCER will take an executable file (COM or EXE), an overlay, a DOS device driver, ROM code in memory, or a code fragment and analyze it using an internal 80X86 simulator and multisegment data and code analyzer. From this analysis, SOURCER produces a clear source code listing in either an ASM source code file or a listing format similar to that produced by an assembler. SOURCER automatically generates the necessary assembler directives, code, and even detailed comments. The comments document program flow, I/O port functions, and DOS and other interrupt functions.

SOURCER supports instruction sets for the 8088/86, 80186, 80286 (both protected

MISOSYS

Programmer's Journal downloads are now available from MISOSYS' forum on Compuserve: GO PCS49

DED86™ Version 2

Powerful features in Version 2!

When you need to travel through your disk drive, why settle for a tool that isolates you from the road? DED86 gives you the direct controls you need to explore your disk. It's a full-screen sector-oriented disk/file editor and a page-oriented memory editor. When you want to "unerase" erased files, DED86's flexible KEEP facility does the job without you fussing over FATs.

- Look by cylinder/head/sector, sector or cluster
- Scan free clusters to search for erased data
- Jump about subdirectories
- Keep sectors & clusters for writing to a file
- Edit bytes in hexadecimal or ASCII, zap in 0s
- Search disk, file, or memory in ASCII or hex
- Touch directory with your date and/or time
- Obtain complete disk statistics in one screen
- Alter attributes: archive, system, hidden, read
- Save/Restore sectors to/from auxiliary buffers
- DOS subshell available while using DED86
- Handles 5.25" & 3.5" drives & RAM disks

RATFOR-86™ 2.0

If you're still fussing with FORTRAN, RATFOR-86 by James Beard, Ph.D., provides structure and greater portability to FORTRAN programs; reduce your programming time and effort dramatically over that required when writing directly in FORTRAN.

- Has extensive macro support
- "arith" macro for binary arithmetic
- 6 new conditional macros in 2.0
- short form READ and PRINT
- keyword "cswitch" like in C
- Requires a FORTRAN compiler

ED/ASM-86™

Integrated Assembler

Editor, assembler, linker, and debugger are integrated into a single .EXE file to ease you into assembly language. Handles 8086, 80186, 80286, and 8087.

- A full featured line EDITOR with intra-line editing, block move/copy, partial save/load.
- A full-screen "Turbo-type" editor included
- ASSEMBLER supports structured coding, can directly generate .COM or .EXE file; assemble to memory for faster debugging.
- The DEBUGGER supports symbolic disassembly of arbitrary code to disk

Purchase now and get a free update to **ED/ASM-386** available soon.

DED86 . . . . . .$59.95
ED/ASM-86 . . . . . .$99.95
RATFOR-86 . . . . . .$99.95
M/C and VISA accepted
S&H: $5, $6 Canada, $15 Other

MISOSYS, Inc.
P.O. Box 239
Sterling, VA 22170-0239
800-MISOSYS or 703-450-4181

Number 121 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

and real mode), V20/V30, 8087, and 80287. These and other options are user controllable. Automatic data type determination includes bytes, words, double words, text, and duplicate bytes. Cross-reference information is embedded as comments in the generating listing file.

In the PC world, disassemblers can trace their heritage back to CP/M-80 and a public domain program (RESOURCE) written by Ward Christenson for interactively disassembled 8080 code. Although interactive disassemblers have traditionally been the norm, SOURCER truly demonstrates the convenience and attraction of an automatic product. Using an interactive product might take hours, days, or even weeks. SOURCER executes fast (typically in minutes) from a full-screen menu system that allows complete control of the process and provides a status report of progress. The program will go through up to five passes building code, data, segments, labels, and comments. Various analysis options selected on screen provide considerable control over the automated process.

However, the normal use of SOURCER still remains a somewhat iterative process except for the simplest of programs. SOURCER's first try will accurately take apart a program, but some areas may be incorrectly determined as data rather than code, or vice versa. Also, labels and subroutine names will be clearly defined, but without symbolic meaning for the programmer. The generated ASM file can still be compiled back to the original executable file, but your understanding of the program flow may be incomplete. However, SOURCER supports a powerful definition file facility which allows complete control of the disassembly process, including the location of segments, code, and data (various types), noting external entry points, and adding descriptive labels and comments. A definition file for a particular program is created with an ASCII editor for use by SOURCER in subsequent runs. I found that with a decent split-screen editor one could review the commented listing output from SOURCER while adding control and label information into a definition file. This definition facility is easily the most flexible and powerful of the three products reviewed here.

A companion product, the BIOS Pre-Processor ($139.95 bundled with SOURCER, $49.95 separate) links with SOURCER to create detailed commented listings from a computer's ROM BIOS. The BIOS Pre-Processor is specifically designed to work with IBM, PS/2, and compatible machines. The preprocessor creates a detailed definition file for your ROM, including comments, external entry points, and labels for key data items.

V Communications has also just released UNPACKER ($39.95), a set of utilities designed to work with SOURCER and assist with software debugging. The primary utility, UNPACK, converts packed EXE files into nonpacked EXE format. (It supports five different packed EXE formats generated by Microsoft.) Other utilities include COM2EXE, which identifies and converts multisegment COM files into EXE files; INTVIEW, an interrupt table viewer that shows all 256 interrupt vectors with comments; and VTEST, which changes video modes and video pages on screen for testing TSR programs. All of these products come with clearly written manuals with indexes—SOURCER, 76 pages; BIOS Pre-Processor, 12 pages; and UNPACKER, 27 pages—and a 30-day, money-back, satisfaction guarantee.

SOURCER could swallow executable files up to about 260K, since the entire executable file is loaded into memory for the process. It handled multisegment code far better than the competitors. Instructions were disassembled correctly even for less-common 286 protected mode instructions. If SOURCER came with an interactive disassembler or could automatically generate definition files, there would be no competition. SOURCER is a polished and professional tool for both the novice and the experienced assembly language programmer.

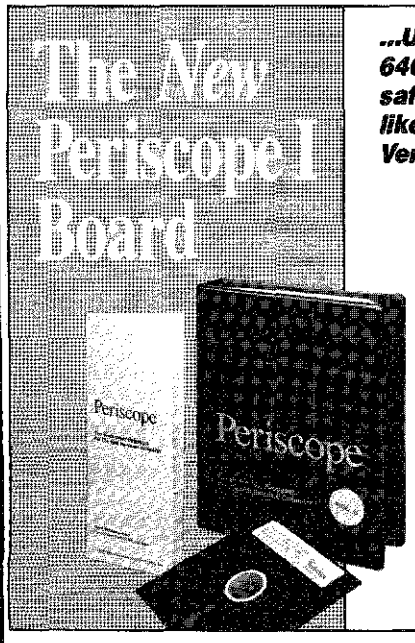
snOOp II
TriDOS Software Publishers
4004 Barbur Boulevard
Portland, OR 97201
(503) 228-8223
$49.95

SnOOp II version 2.01 is an interactive disassembler for the 80X86 family. SnOOp works primarily in the interactive mode, although it can function as a limited batch disassembler as well. SnOOp is an outgrowth of the family of CP/M interactive disassemblers and follows their philosophy with a number of improvements. The snOOp display emulates a word processor containing a source version of the file being disassembled. Memory addresses are on the far left, followed by assembly language opcodes or data, with comments on the right. You can page up or down through the file just as you would with a word processor.

SnOOp supports three file formats: COM, EXE, and SYS (assumed to ORG at 0). The file is never actually disassembled until you choose to produce an ASM file. Instead, snOOp maintains an extensive database in memory on the file being disassembled, including the locations of symbols, comments, data types, and default segments.

The interactive process consists of using the snOOp edit mode to browse through a file. As you browse, you can set data types and segment values, define symbols, and add comments. You write labels to the left of opcodes or data and type comments on the right, as when using a screen editor. Labels are limited to 12 characters; however, case is significant. Multiline comments are supported, albeit a bit awkwardly.

SnOOp's editing commands and cursor movement are generally similar to



...Uses ZERO memory in the lower 640K. Yet it has plenty of room to safely store all debugging information, like symbols, as well as the powerful Version 4 software.

New in Periscope Version 4:

- View local symbols from Microsoft C
- Debug Microsoft Windows applications
- Set breakpoints in PLINK overlays
- Improved source-level support
- Monitor variables in a Watch window
- 80386 debug register support
- Debug using a dumb terminal
- PS/2 watchdog timer support
- Use mixed-case symbols

"The best keeps getting better... Once again, Periscope has raised the industry standard for debuggers!"

—David Nanian
BRIEF co-author

Periscope I includes a NEW full-length board with 512K of write-protected RAM; break-out switch; software and manual for $795.

Periscope II includes break-out switch; software and manual for $175.

Periscope II-X includes software and manual (no hardware) for $145.

Periscope III includes a full-length board with 64K of write-protected RAM, hardware breakpoints and real-time trace buffer; break-out switch; software and manual. Periscope III for machines running up to 10 MHz is $1395. Plus the new Model I board, $1995.

Prices subject to change without notice.

Call Toll-Free for free information or to order your Periscope today!

MAJOR CREDIT CARDS ACCEPTED. **800-722-7006**

The Periscope Company, Inc.
1197 PEACHTREE ST.
PLAZA LEVEL
ATLANTA, GA 30361
404/875-8080

Number 6 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

The text and table at the right are from "Migrating to Graphics Processors," Jake Richter's "On Graphics" article in *PJ* 7.1. The material was supposed to appear in the first column on page 38 of that issue. We apologize for the inconvenience.

Wordstar's. Saving a database file or exiting is accomplished with the Wordstar Ctrl-K commands. The ten function keys are used effectively for help and to set the data type, assign a symbol, set 80X86 segments, jump to a location, and find a string of bytes in the file. The data type may be set as code, a procedure start or end, ASCII text, data bytes, words, or double words. A reverse Polish notation calculator is also available via a function key, as is a series of automatic option controls.

The program supports an automatic concurrency option that builds labels and controls entries and comments in the background while waiting for keyboard input. This background concurrency feature operates very slowly; so it should not be compared with automatic products like SOURCER. Because snOOp does not emulate the 80X86 registers, comments are generic to the instruction and not specific to the disassembled program.

Several features stand out about this product. SnOOp comes with an extensive online help database on opcodes, DOS function calls, BIOS video functions, Mouse functions, EEMS functions, and Topview calls. This information is available via function key help. The edit mode also supports jump-to-location and return options that are stack based (that is, each jump location is pushed on a stack), making it convenient for exploring a subroutine and then backtracking. SnOOp keeps the entire database in memory, but the program being disassembled is paged in and out. All available memory is used. With this design, even large files may be partially disassembled. SnOOp II comes with a 45-page manual with an index.

To use this product requires experience with 80X86 assembly language, since the user makes most of the decisions. I found the concurrency option would generate too many spurious labels in the wrong segments; they were more trouble to delete. The manual states that snOOp will recognize instructions from the entire 8086/8087 through 80386/387 family; however, my experience indicates only 8086 code is fully recognized. New instructions added with the 80186 were disassembled as data bytes instead of instructions, although the help database had the correct information.

SnOOp did not handle multisegment programs easily or well. I couldn't find a way to force a MOV immediate (MOV AX, 1234H) to be interpreted as a label offset. The output file can be either an ASM source file or a MAP file. Except for very small and simple files, generating an accurate ASM source file is labor intensive. However, the

NEW VERSION 4.0

DYNAMITE DEMOS

Proteus Prototype/Demo System

Why fizzle with an outdated demo program when you can explode with Proteus?

Proteus is the first prototype/demo system to combine complete simulation capability, a powerful Screen Painter, video effects, custom sound and much more in a program you don't have to be a genius to figure out.

And now Proteus supports all graphic modes for the CGA, EGA and VGA in the popular PCX file format.

"A joy to use."
. . . Computer Language

- **Complete Simulation Capability**
- **Powerful Screen Painter**
- **Bit-Mapped Graphics**
- **Video Effects**
- **Custom Sound**

Don't Let Your Next Demo be a Dud!
Find out why software developers from Ashton-Tate to ZyLogic have switched to Proteus.

1-800-634-9986

$149. *No royalties. 30-day Money Back Guarantee. MC/VISA/COD/PO accepted.*

Helios

P.O. BOX 22869, SEATTLE, WA 98122

Number 258 on the Reader Service Card

A GP command sequence is like a program, but in a very simple language. You load a string of GP commands into the 82786 memory, terminate it with an NOP that has its ECL (End of Command Line) bit set, and then tell the GP to start executing from a specific point in memory. Since the chip does not have any conditional branch instructions, its

Table 1. Logical Writing/Pixel Processing Mode on the 34010 (The first 16 are also supported by the 82786.)

Definitions:

0s	All bits in the pixel are set to 0
1s	All bits in the pixel are set to 1
DST	The pixel at the destination
SRC	The source pixel
AND	Bitwise AND operation
OR	Bitwise OR operation
XOR	Bitwise XOR operation
NOT	Bitwise NOT (one's complement) operation
ADD	Unsigned Addition
SUB	Unsigned Subtraction
ADDS	ADD with Saturation (no wrap max's out at max pixel value)
SUBS	SUB with Saturation (no wrap, stops at a pixel value of 0)
MAX	Larger of the two operands
MIN	Smaller of the two operands

The result of a pixel processing operation is placed in DST.

Processing Modes:

```
SRC
SRC AND DST
SRC AND (NOT DST)
0s
SRC OR (NOT DST)
(NOT SRC) XOR DST
NOT DST
(NOT SRC) OR (NOT DST)
SRC OR DST
DST
SRC XOR DST
(NOT SRC) AND DST
1s
(NOT SRC) OR DST
(NOT SRC) AND (NOT DST)
NOT SRC
SRC ADD DST
SRC ADDS DST
DST SUB SRC
DST SUBS SRC
MAX(SRC, DST)
MIN(SRC, DST)
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

MAP file from a partial disassembly could be useful with a symbolic debugger. SnOOp's weaknesses are countered by the extensive opcode help feature and large file capability.

MD86
C. C. Software
1907 Alvarado Avenue
Walnut Creek, CA 94596
(415) 939-8153
$67.50

C. C. Software has released version 2.1 of MD86 (Masterful Disassembler) for the PC. The company previously developed disassemblers for CP/M as well as source code generators for Turbo Pascal v3.0 both on the PC and under CP/M. MD86 is their version of an interactive disassembler for the 80X86. MD86 is similar to the snOOp product in both philosophy and implementation. MD86 provides a full-screen environment to disassemble an executable file, a BIOS ROM, or a code fragment. The program uses the PC's arrow, page-up, page-down, end, and home keys for browsing through the file. Function keys are used to set data types, to define labels, for searching, for help, and for other purposes. Function keys also support a stack-based jump-and-return mechanism similar to snOOp's. Various options can also be set through these function keys and exploding pop-up menus. Global options control the starting and ending address, the instruction set supported, values for the segment registers, whether numeric coprocessor instructions are decoded, automatic commenting, and default options for handling MOV immediate instructions.

A pop-up calculator is available, along with on-line help for MD86 function keys, DOS functions, and PC ROM BIOS and DOS interrupts. The on-line help database is not so extensive as the one in snOOp. The calculator supports both hexadecimal and decimal integer arithmetic and bitwise operators with results displayed as 32-bit signed integer values (both hex and decimal). One can scan an entire file and build the label address pool. Labels are limited to eight characters but are case sensitive. Multiline comments are well supported. Data type specifications are limited to ASCII text, data bytes, words, and uninitialized data. MD86 comes with a 40-page manual (no index).

MD86 currently runs in 128K and uses a single 64K data segment, which limits the size for the address symbol, comment, and data type entries. This restricts its use in disassembling large files without breaking them into pieces. A maximum of 128 segments are supported. MD86 will actually handle large files reasonably well aside from these table limits. I found that MD86 properly decodes most but not all 80286 instructions encountered. Also, a line attribute option allows forcing a MOV immediate to reference a label in a particular segment. I preferred the snOOp user interface implementation, but MD86 did a better job disassembling the sample code I tried.

Books

The 80386 Book **by Ross P. Nelson**
Microsoft Press
16011 N. E. 36th
Redmond, WA 98073
(206) 882-8080
$24.95

Although several dozen books have been released on the 80386, this new title comes closest to being a solid introduction and reference work. Ross Nelson, a former Intel employee, offers assembly language programmers and others interested in computer architecture a clear, comprehensive, and authoritative introduction to the 80386. This book dwells solely on the software and programming aspects of the 80836/80X87 and does not cover hardware issues. The main focus of the text is on system programming in the protected mode (often called native mode) of the 80386. The first 150+ pages present a clear description of the CPU architecture, memory segmentation, protection mechanism, paging, and 80287/80387 programming. The next 230 pages are the required 80386/80387 instruction set reference. The final 60+ pages are appendices, including various opcode tables, instruction format and timing tables, instruction disassembly tables, and notes on 80X86 processor differences.

Chapters cover 386 memory architecture—using segmentation and paging, the 80386 protection mechanism, the basic instruction set, and real and virtual 8086 modes. If you want a readable understanding of selectors, privilege, interlevel communication, task switching, interrupts, and exceptions, this book will provide it. Clear, detailed descriptions are provided of task state segments (TSS), the various descriptor formats, I/O permission bit maps, page tables and directories, the translation lookaside buffer, page faults, and call, interrupt, and task gates. The 80386 instruction set reference is ample and notes exceptions that may occur in either real mode, protected mode, or virtual 8086 mode. This reference section could be strengthened, however, with more examples and elaboration.

Be forewarned that this book does not contain assembly language programming examples for setting up any of these protected mode structures. You'll have to refer to the Intel 80386 reference manuals for such examples. Also, the Nelson book omits any coverage of the Weitek math coprocessor. For both 387 and Weitek coverage, I continue to recommend *The 80386/387 Architecture* by Stephen Morse, Eric Isaacson, and Douglas Albert (also former Intel folks) from Wiley (1987). However, the description and explanations of the 80386 protection mechanisms in *The 80386 Book* are clearly organized and well written. Complemented with the Morse book and Intel manuals, *The 80836 Book* makes an excellent programmer's reference. ♦

QEdit *Advanced*

the "quick editor"

By SEMWARE

QEdit is one of the smallest, fastest, most easy to use multi-file editors available for IBM Compatible Computers with many features to enhance your productivity.

- *Extremely Fast*
- *Completely Configurable (Including Keyboard!)*
- *Easy to Use Macro Facility*
- *Optional "Pop Down" Menu System*
- *Recover Deleted Text*
- *Edit Dozens of Files Simultaneously*
- *Utilize up to 8 Windows to View Files*

QEdit *Advanced* requires an IBM PC/XT/AT or Compatible, IBM PS/2 30-80, 128KB of available memory.
Please specify 5.25″ or 3.5″ diskette.

$54.95 MASTERCARD & VISA ACCEPTED
Add $3.00 Shipping & Handling. Georgia Residents Include 4% Sales Tax.

To Order:
SEMWARE, 730 Elk Cove Court
Kennesaw, Georgia 30144
Phone Orders: (404) 428-6416

Number **148** on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

C SOURCE for BUSINESS and ACCOUNTING that TURNS LITTLE LEAGUE into GIANTS

What seperates sand-lot from Super League?

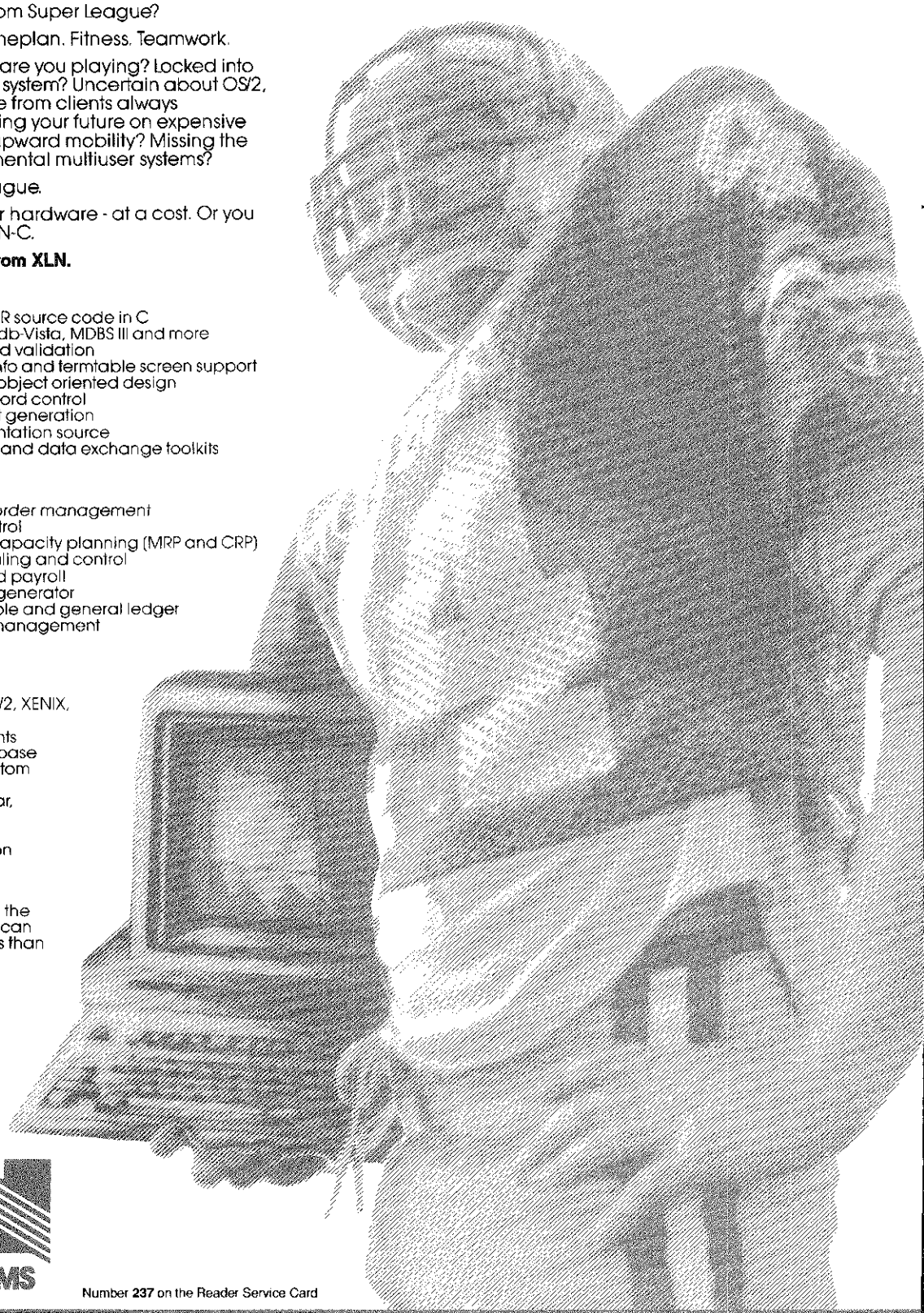
Top Coaches. A better gameplan. Fitness. Teamwork.

So what kind of ball game are you playing? Locked into your proprietory database system? Uncertain about OS/2, UNIX, LAN's? Under pressure from clients always demanding more? Gambling your future on expensive 4GL's? No data sharing or upward mobility? Missing the massive growth in departmental multiuser systems?

That's no way to win the league.

You could standardise your hardware - at a cost. Or you could switch the play to XLN-C.

Buy your C learning curve from XLN.



The skills

- Complete documented K & R source code in C
- Interfaces to c-tree, C-ISAM, db-Vista, MDBS III and more
- Sophisticated data entry and validation
- Video, BIOS, termcap, terminfo and termtable screen support
- Fast, portable and efficient object oriented design
- 1-2-3 style menus with password control
- WYSIWYG screen and report generation
- TEX typesetter and documentation source
- High level graphics, comms and data exchange toolkits
- LISP and Prolog AI system

The plays

- Sales, purchase and works order management
- Multi-location inventory control
- Distribution, materials and capacity planning (MRP and CRP)
- Critical path activity scheduling and control
- Personnel management and payroll
- SQL database application generator
- Accounts receivable, payable and general ledger
- Maintenance and service management
- Fixed asset management

The touchdowns

- No runtime royalties
- Identical source for DOS, OS/2, XENIX, UNIX and VMS
- Portable to 60 + environments
- Upwardly compatible database
- Flexible source code for custom applications
- Grow with standard, modular, integrated object design
- Fast, portable, multiuser, multitasking business solution
- Unlimited technical support

Gate money

$395 gets you in the gates with the module of your choice: or you can cover the whole season for less than the price of a PC.

XLN turns you into a master strategist: while C turns your cowboys into Cowboys.

EXCELENT SYSTEMS

Number **237** on the Reader Service Card

XLN SOFTWARE 3680 Tuggle Road, Buford, GA 30518. Phone orders: 1 (800) 543-5770. Fax: (404) 859-0051. Georgia, International and Support: (404) 945-874

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

UNDERSTANDING C VIDEO TAPE

Get guaranteed results with this three volume video course! For expert or novice—these tapes (total 6 hours) cover ANSI C for today's professional programmer. Loaded with examples, these tapes cover it all! Handy reference book included.

$59.95 postpaid. Add appropriate tax in TX. Send check or money order to:

AUTOMATION EDUCATION
2309 Royce Drive
Arlington, TX 76016-1232
817-496-8747 **RS# 207**

AT LAST! A C SOURCE CODE ANALYSIS TOOL WITH FULL PREPROCESSING CAPABILITY

C:LINES/C:TREE features

- ANSI draft compatible preprocessor
- Outlined listings showing flow of control within procedures
- Tree diagrams showing flow of control between procedures
- Source code reformatter

$79.95, shipping $3 domestic, $15 overseas. MD residents add 5% sales tax. Requires PC compatible, 256K.

SoftRex
4807 Bethesda Ave. #287
Bethesda, MD 20814
301-881-8274 **RS# 185**

BASIC to C

BAS C v3.2 accepts BASICA, Quick BASIC, CBASIC86, or most other BASICs by adding new statements, functions(Language Extender). Generates structured, indented, scoped MS/Turbo/Lattice/Aztec C. All Memory model, compile option supported. Runs on MS-DOS, XENIX, UNIX. Demo disk. From $199.

GOTOLESS CONVERSION
P.O. Box 835910
Richardson, TX 75083
214-404-1404 **RS# 127**

PL/M TO C TRANSLATOR

LISP-based translator for major assistance in translating from INTEL PL/M to C language. Human readability a high priority. Sample listings available. Runs under MS-DOS 2 and up. Ports possible for VAX with VMS or BSD 4.2.

$5000 site license includes 3 months of telephone support.

Frank Korzeniewski Consulting
1564-A Fitzgerald Drive #137
Pinole, CA 94564
415-799-1819 **RS# 209**

Computer & Technical
*** BOOKS ***
INFOSOURCE
International
PROGRAMMERS PC SOURCEBOOK $24.95
TURBO C DEVELOPERS LIBRARY $24.95
Box 288, Dept. PJ8
Barryville, NY 12719-0238
1-800-327-7970

Number 217 on the Reader Service Card

LATEST AWARD BIOS

Update Your PC/XT • 286 • 386
Support for:
- **Enhanced Keyboards**
- **EGA & VGA Graphics**
- **Custom Drive Tables**
- **3.5" Floppies & More**

Authorized AWARD Distributor
(800) 423-3400
KOMPUTERWERK, INC.
851 Parkview Blvd.
Pittsburgh, PA 15215

Number 155 on the Reader Service Card

D-ISAM

Industry standard file manager. Only $145 for DOS libraries, $595 for UNIX/DOS source code.

'W'

UNIX/DOS character windowing. $95 for DOS libraries, $295 for UNIX/DOS source code.

BYTE DESIGNS LTD.
#6-3511 Viking Way
Richmond, B.C.
V6V 1W1 CANADA
(800) 663-8547
(604) 278-5200

Number 219 on the Reader Service Card

Turbo Taskforce.

Want to write better code in less time? We're the Turbo User Group, and since 1984 we've been helping people do that and more with all of Borland's language products. Membership in TUG will net you -

- A subscription to TUG Lines, the group's bi-monthly journal, containing utilities, bug reports, patches, workarounds, product reviews and more
- Access to TUG's "Troubleshooters" - volunteers who help solve programming problems by phone
- A discount on The TUG Boat, the group's multi-user BBS, where you can contact, correspond and conference with fellow users, participate in language SIGs and download "hot" Public Domain software
- Access to TUG's extensive disk library of hand-selected and tested Public Domain and ShareWare utilities, routines and applications
- And more!

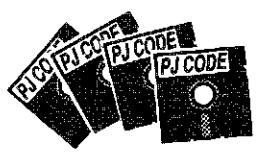
Don't just work *harder* - join with us in our commitment to work *smarter!*

Turbo User Group
Post Office Box 1510 • Poulsbo, Washington 98370
(206) 779-9508

Annual dues (US funds): U.S. $24.00; Canada $28.00; Overseas $39.00

Number 259 on the Reader Service Card

PJ Code Is Available!



On disk from:

Programmer's Journal
P. O. Box 30160
Eugene, OR 97403
(800) 234-0386
(503) 747-0800 in Oregon

All Code Disks $10.00 each.
To order see ad on page 88.

PJ Code DiskScriptions are available for only $39.95.
To order see ad on page 8.

On BBS from:

Mnematics Videotex
P.O. Box 19
Sparkill, NY 10976-0019
Voice (800) 322-3633
Voice (914) 365-0184 in NY
Mention ***PJ*** when you call.

FIDO/OPUS Network
Blazer Net 152 Node 9
(503) 343-9724

Generation Five BBS
Voice (301) 495-6690
Data (301) 495-2932
Accessible via PC Pursuit, Washington D.C. From Files submenu, select P for Personal Downloads.

PCS-49 on CompuServe
Section 9
Sponsored by:
MISOSYS, Inc.
P.O. Box 239
Sterling, VA 22170
Voice (800) MISOSYS

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

ADVERTISER INDEX

Reader Service	Advertiser Name	Page Number
220	Aesoft Corporation	58
215	American Cosmotron, The	87
30	Austin Code Works, The	65
207	Automation Education	95
51	Az-Tech Software, Inc.	26
244	Block Island Technologies	45
265	Bolt Systems	45
3	Borland International	C4
219	Byte Designs, Ltd.	95
263	Cater Software	63
255	C-Team, The	16
229	Disk Software, Inc.	22
241	Emerald City Software	8
243	English Knowledge Systems	66
10	Essential Software	60
209	Frank Korzeniewski Consulting	95
261	Genus Microprogramming	22
*	Gimpel Software	52
127	Gotoless Conversion	95
204	Guidelines Software, Inc.	53
248	Hammerly Computer Services, Inc.	C3
122	Hauppauge Computer Works, Inc.	C2
154	HavenTree Software Ltd.	64
258	Helios Software	92
217	Infosource International	95
239	INTEK, Integration Technologies, Inc.	55
58	Interface Group, The	43
226	Ithaca Street Software, Inc.	25
211	KADAK Products Ltd.	15
155	Komputerwerk	95
63	Laboratory Microsystems, Inc.	83
44	Lahey Computer Systems, Inc.	24
246	Logical Alternatives, Inc.	10
212	Magna Carta Software	54
233	Mansfield Software Group	14
223	Marietta Systems	67
143	Metagraphics Software Corporation	23
45	MetaWare, Inc.	1
232	Microsoft Corporation–Human Resources	9
234	Microsoft Press	17
240	Milton Brown Software	85
121	MISOSYS Inc.	90
238	MISOSYS Inc. Game Division	16
94	MMC AD Systems	67
109	NANOSOFT	47
160	Nu-Mega Technologies	3
16	Opt-Tech Data Processing	50
254	OPUS Software	15
195	Paradigm Systems	83
80	Paul Mace Software	30
208	PC Tech, Inc.	27
6	Periscope Company, The	91
*	*PJ Code DiskScription*	8
*	*Programmer's Journal* Mail Order Store	88
228	ProTech Marketing, Inc.	12
97	Qualstar Corporation	83
252	Quantum Software	26
251	Radix Systems	14
222	Rainbow Technologies	15
20	Rational Systems, Inc.	51
198	Research Group, The	13
148	SEMWARE	93
165	SLR Systems	7
33	Soft Advances	86
185	SoftRex	95
98	Software Engineering Store, The	35
221	Software Security	5
257	Sophisticated Software, Inc.	81
158	Summit Information Systems, Inc.	77
85	Sunny Hill Software	77
253	Sutrasoft	21
264	TECHSEL	16
166	Tool Makers, The	62
262	Trio Systems	61
25	TurboPower Software	85
259	Turbo User Group	95
125	V Communications	89
100	Western Wares	67
73	Whitewater Group, The	59
237	XLN Software	94

When contacting any of the above advertisers, please tell them you saw their ad in ***Programmer's Journal*** and that you appreciate their continuing support of this magazine. The advertisers with an * prefer to be contacted directly. ***PJ***'s Advertiser Index is provided as a free service, and ***Programmer's Journal*** cannot be held responsible for omissions or errors.

COMING ISSUES

May/June

Database Programming:

SQL *by Richard Riehle*

Database Normalization *by Bill Madison*

On Graphics:

Bresenham's Algorithm *by Michael Abrash*

PJ Forum:

Neural Nets *by Tom Waite and Hal Hardenbergh*

July/August

Networking and Communications

Advertise in *Programmer's Journal*

May/June 1989 Volume 7.3

Space Reservation Deadline:

March 10, 1989

Materials Due:

March 24, 1989

Please call *Programmer's Journal* to reserve your space.

(503) 747-0800

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Six Five easy ways to boost your BASIC

PROBAS™ TOOLKIT

The TOOLKIT is a collection of assembly and BASIC modules that use the PROBAS library to save you even more hours of grunt work. Why spend hundreds of hours re-inventing the wheel when you can just plug in TOOLKIT modules like:

- **Menu Generators**
- **Fast B-tree indexing**
- **Mini-editor with word-wrap**
- **Patch .EXE files**
- **Protected storage areas**
- **Julian date routines**

The TOOLKIT also includes clock, calendar, windowing, BCD math routines and much more, complete with BASIC source code and a comprehensive manual. The PROBAS TOOLKIT adds capabilities and helps conserve your most valuable asset of all—time! Requires PROBAS. ***Just $99.00!***

PROBAS™ TELECOMM TOOLKIT

The PROBAS TELECOMM TOOLKIT is a collection of high-level communications modules that you plug into your code to provide popular file transfer protocols, terminal emulations, login scripts and baud rates up to 115,200 baud. You get:

- **Xmodem/Modem7/Xmodem-1k**
- **Ymodem (single and batch)**
- **CRC-16 and Checksum**
- **VT52, VT100, ANSI BBS etc.**
- **Auto Dialer & data base**
- **Documented BASIC source**

Why use clumsy SHELLs to complex terminal programs when you can plug just the communications routines you need into your code? Implement just the features and commands you want. Requires PROBAS. ***Just $75.00!***

Our thirty day, money-back guarantee assures you the highest quality and our technical support staff is always ready to help. Try our BBS at **(301) 953-7738** or give us a call at:

HAMMERLY
COMPUTER SERVICES, INC.
8008 SANDY SPRING ROAD • LAUREL, MD 20707

(800) 343-7484

Add $5.00 per item ($8.00 Canada) for shipping per order. Europe: $39.00 for 1st item plus $5.00 for each additional item. Visa, M/C, C.O.D. (US Only) checks and approved POs accepted. Trademarks PROBAS, PROREF, PROSCREEN, PROMATH: Hammerly Computer Services, Inc. QUICK BASIC, BASCOM: Microsoft Corp.

Number 248 on the Reader Service Card

PROBAS™ updated

Professional Basic Programming Library

Announcing PROBAS Version 3.0, now with over 335 assembly routines to really kick QuickBASIC and BASCOM into high gear. BYTE magazine calls PROBAS a "Supercharger for QuickBASIC". Thousands of programmers rely on PROBAS to make their life easier and to enhance their programs with features like:

- **An 800-page 3-part manual**
- **Full-featured windowing**
- **Screen snapshots (Text & Graphics)**
- **String, array, and pointer sorts**
- **Lightning-fast file I/O**
- **Full mouse support**

Create dazzling screens in text mode, CGA, EGA, VGA or Hercules graphic modes. Save and restore screen snapshots to arrays, EMS memory or files. Full featured windowing to meet the most demanding jobs. The PROBAS system of virtual screens allows you to draw full or partial screens to memory, and then snap them on in an eyeblink. You can even create vitural screens far larger than the display screen.

Sick of running out of string space? Store hundreds of K in numeric arrays or megabytes in extended or expanded memory. Tired of using a kludgy SHELL to DIR to read a directory or archive files? Scan subdirectories or .ARC files using wild-cards and store thousands of file names, dates, and times. Wish you could drag a window containing text or a menu around the screen with a mouse? It's easy!

PROBAS gives you a complete set of blazingly-fast file routines. Read or write huge chunks of data at a clip, with file locking and error handling so that you can even use them in subprograms. You'll never want to use BASIC's file I/O again! Sort data with lightning fast array and pointer sorts. Search files or arrays at assembly speeds. PROBAS also has over 200 other essential services including handy string, date, time, directory and array manipulation, string, screen and data compression, full mouse support, valuable equipment and input routines and faster replacements for most BASIC commands.

Whether you are a professional or a novice, PROBAS will boost your BASIC in ways you never dreamt possible. PROBAS allows professionals to save time and work and lets novices write professional-quality programs quickly and easily. After all, how much is a few hundred hours of your time really worth?

For all versions of QuickBASIC and BASCOM including BASCOM 6.0 for OS/2. ***Just $135.00!***

PROREF™

On-Line Help For PROBAS

PROREF provides pop-up help for the routines in PROBAS and is an extension of the QuickBASIC programming environment. Find help on any routine with a few keystrokes or mouse clicks. Pop-up an ASCII chart, calculator, scan code module, box diagram, your own help information or almost any DOS program via a hot-key. ***Just $50.00!***

PROSCREEN™

Professional Screen Management System

PROSCREEN is a full-featured screen generator/editor that will save you more design and coding time than you ever thought possible. PROSCREEN treats screens like a word processor treats text to provide complete control over characters, colors, and placement. Design input screens with up to 130 fields and 19 pre-defined and 2 user-defined masks. Use PROBAS or the included BASIC/Assembler subroutines to access the screens. No kludgy code generators here! Comes with subroutine source, extensive on-line help, and a 285 page manual. ***Just $99.00!***

PROMATH™ new!

PROMATH is a collection of over 150 high-level routines that provide mathematical functions and operations for programmers who often work in mathematics, science, or engineering. Complex variables, real and complex matrices, real and complex trigonometric and hyperbolic functions and their inverses, solution of linear equations, integration, differential equations, Fast Fourier transforms and many other useful routines are provided.

For years Fortran has been the language of choice for scientific and engineering applications, but it lacks many of the useful features of QuickBASIC. PROMATH contains most of the Fortran mathematical and numeric functions and allows you to easily translate Fortran code to BASIC or write new programs in BASIC while retaining Fortran's numerical prowess.

The PROMATH manual is over 200 pages and provides a complete description of each routine, including any algorithm and the mathematical formula the routine uses, shown in standard notation. For QuickBASIC 4 and BASCOM 6 only. ***Just $99.00!***

PRODUCED 2004 BY CMZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

NEW TURBO DEBUGGER

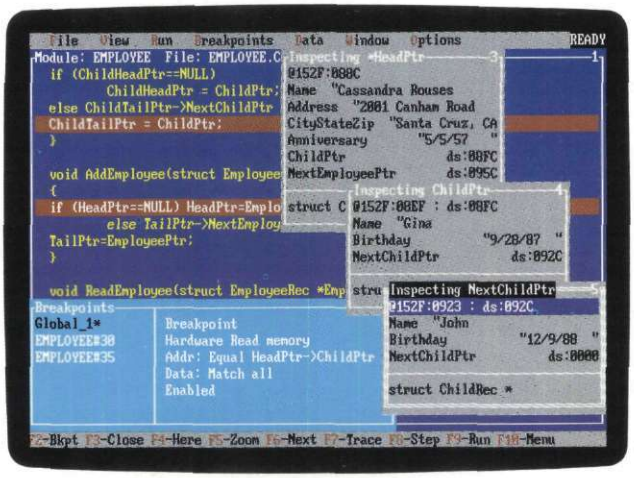
Change your views.

Getting 12 hours of work done in 8 with our new Turbo Debugger® is the kind of accomplishment that will instantly change your views about debuggers and debugging.

So will the ability to debug any size program—because with Turbo Debugger you can move into the virtual mode of the 386 and use up virtually no memory. You can even debug remotely on a different machine.

You see 12 *different* views of your code. You see *everything* and you see it now.

These are just a few of the features of our new Turbo Debugger. It's the new power tool that every professional programmer needs now if your debugging is to be state-of-the-art instead of primitive art.



Data debugging: An easy way to browse through your data structures.

It's easy to change your views

Switching from an older debugger to our new Turbo Debugger is painless. If you're currently using another debugger or thinking about buying one, think twice.

Turbo Debugger can easily convert CodeView® programs to make your debugging easier, smoother, and a lot more efficient in every way.

To point is the point

Point at something in Turbo Debugger and you instantly see what you want to see—and if you can see it, you can change it. Pointing is the high-tech simplification of a great deal of sophistication.

Turbo Debugger's menu-driven commands are a lot more advanced, logical, and productive. You're in an easy intuitive environment that's fast, powerful, and professional.

You get all our support when you change your views

CompuServe,® User Groups, and our own Technical Support experts are all ready to help you when you change your views. The Turbo Debugger manual is clear, communicative, and comprehensive.

60-day money-back guarantee*

*Customer satisfaction is our main concern; if within 60 days of purchase this product does not perform in accordance with our claims, call our customer service department, and we will arrange a refund. All Borland products are trademarks or registered trademarks of Borland International, Inc. CodeView is a registered trademarks of Microsoft Corp. Copyright ©1989 Borland International, Inc. BI-1310

Code: MD02

Number 3 on the Reader Service Card

Critically acclaimed!

New Turbo Debugger wins PC Magazine's 1988 award for "Technical Excellence."

"Everyone who's tried the Turbo Debugger agrees: It wins the (development tool) category's award for Technical Excellence hands down. The user interface is simple yet elegant; the program works the way programmers want to work. Yet again, Borland has advanced the state of the art in an eminently useful way."

Bill Machrone, *PC Magazine*

"CodeView takes up too much room. Turbo Debugger can go into the virtual mode of the 386, taking up almost no memory."

Bruce Lynch, *PC Week*

TECHNICAL EXCELLENCE • PC MAGAZINE

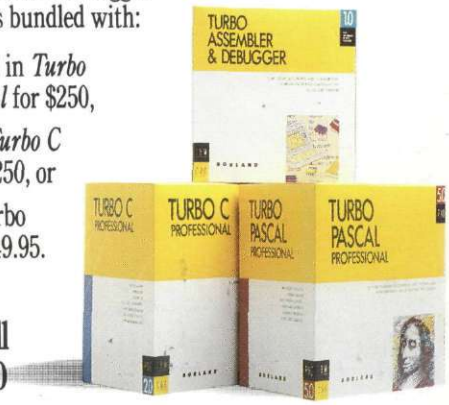
"In a random survey of programmers familiar with both tools, most agree that Borland's new entry outperforms the Microsoft product hands down."

Ann Sussman, *PC Week*

3 different ways to change your views.

Turbo Debugger comes bundled with:

- Turbo Pascal® 5.0 in *Turbo Pascal Professional* for $250,
- Turbo C® 2.0 in *Turbo C Professional* for $250, or
- With our new Turbo Assembler for $149.95.



For the dealer nearest you, call (408) 438-5300

BORLAND

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED